Escuela
Politécnica
Superior

# Generación de datos sintéticos para segmentación de acciones en secuencias de vídeo

# Fin de Grado

Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor:
Plácido Antonio López Ávila
Tutor/es:
José García Rodríguez, Pablo Martínez González
y John Alejandro Castro Vargas

Junio 2019

Universitat d'Alacant
Universidad de Alicante

UNIVERSIDAD DE ALICANTE

TRABAJO FINAL DE GRADO

# Generación de datos sintéticos para segmentación de acciones en secuencias de vídeo

*Autor*
Placido Antonio LOPEZ AVILA

*Supervisor*
Jose GARCIA-RODRIGUEZ

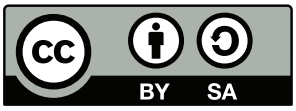
*Una tesis presentada en cumplimiento de los requisitos para el grado de Ingenieria Informatica*

Grado en Ingenieria Informatica
Departamento de Tecnología Informática y Computación

3 de junio de 2019

Este documento ha sido realizado con el apoyo de LaTeXy Ti*k*Z.





*«Dentro de tres semanas yo estaré recogiendo mis cosechas, imaginad donde querréis estar, y se hará realidad. Manteneos firmes, no os separéis de mi. Si os veis cabalgando solos por verdes prados, el rostro bañado por el sol... Que no os cause temor, estaréis en el elíseo, y ya habréis muerto. Lo que hacemos en la vida, tiene su eco en la eternidad.»*

Máximo Décimo Meridio (Gladiator)

# Resumen

El objetivo principal de este trabajo es la generación sintética de datos para comportamientos en tercera persona, incluyendo técnicas de deep learning para procesar los datos y obtener predicciones y conductas. Para el desempeño de esta propuesta, ha sido necesario obtener representaciones de las acciones más comunes del día a día en el hogar, procesarlas para generar un etiquetado acorde a estas y emplear **Redes Neuronales Convolucionales** para el post-procesado. Junto con este desarrollo se ha trabajado con tecnologías como **Unreal Engine 4** y **PyTorch**. Respectivamente, el primero se ha empleado para cargar datos pre-grabados con el traje **AxisNeuron** y generar una volumetría de datos mucho más grande para después obtener estos vaticinios por medio de del framework nombrado en segundo lugar. La experimentación incluye múltiples análisis y pruebas que se han generado de cada una de las dos partes en las que se puede dividir el proyecto: extracción de datos y procesamiento de los mismos. Como resultado, se ha obtenido un sistema capaz de observar y obtener pronósticos en un rango de acciones segmentadas en tercera persona dentro de las acciones más comunes en el hogar.

# Agradecimientos

Dedico todo mi esfuerzo a mis padres, quienes nunca han dejado de confiar en mi. Y especialmente a mi hermano, a quien espero que este trabajo le sirva de motivación para llegar muy lejos.

Por supuesto, no puedo olvidar al apoyo que he recibido tanto por mi tutor Jose García Rodríguez, como por los compañeros del departamento. En especial Pablo Martínez Rodríguez y John Alejandro Castro Vargas, quienes compartieron conmigo cada uno de sus conocimientos y me brindaron su ayuda a cada instante.

# Índice general

# Índice de figuras

# Lista de acrónimos

**3D** Entornos Tridimensionale

**PRAC** Probabilistic Robot Action Cores

**KB** Conjunto del conocimiento base para un ámbito en concreto

**RGB** Modelo aditivo de colores Rojo, Verde, Azul

**DTIC** Departamento de Tecnología Informática y Computación de la Universidad de Alicante

**BVH** Biovision Hierarchy

**FBX** Filmbox

**PC** Ordenador Personal

**GPU** Unidad de Procesamiento Gráfico

**FPS** Fotogramas por segundo

**JSON** Notación de objeto JavaScript

**UDP** Protocolo de datagramas de usuario

**TCP** Protocolo de control de transmisión

**MIT** Instituto de Tecnología de Massachusetts

**SVG** Gráficos de visualización semántica

# Capítulo 1

# Introducción

*En el siguiente capítulo se va a mostrar una vista general sobre el proyecto desarrollado. Este está organizado la se siguiente forma. Sección 1.1 con un planteamiento general de los problemas tratados en el plan de trabajo. Sección 1.2 indicando la motivación que inició el estudio de este rompecabezas. Sección 1.3 donde se plantean trabajos relacionados, que han servido de inspiración para desarrollar un esquema inicial sobre el que comenzar a trabajar.*

## 1.1. Vista general

En el siguiente proyecto de final de grado, voy a exponer una serie de problemas relacionados con la generación de datasets sintéticos orientados al reconocimiento de acciones en el hogar/entornos familiares, lo que sea... Por otro lado, analizaremos diferentes técnicas utilizadas para reconocimiento de acciones con el objetivo de definir las características de un dataset útil y robusto para dicha tarea. Junto a ello se plantearan las soluciones ideadas y desarrolladas para llevar a cabo la resolución de esta serie de problemas. Las técnicas basadas en deep learning han permitido superar los resultados obtenidos por técnicas tradicionales. Sin embargo, todas ellas necesitan nutrirse de un gran cantidad de datos, que a su vez necesitan tener la mejor calidad posible para realizar un correcto entrenamiento de los diferentes algoritmos o redes neuronales. Finalmente se presentan análisis de rendimientos y pruebas con las diferentes configuraciones empleadas.

## 1.2. Motivación

Durante mi adolescencia siempre he tenido claro hacia donde iba a encaminar mis pasos y estos me llevaron a cursar el *Grado de Ingeniería Informática* en la *Universidad de Alicante*. Llegado a este punto, los caminos que tomar eran tan numerosos que era casi imposible decantarse por uno solo. Pero esto cambió a partir de la segunda mitad del grado, cuando la inteligencia artificial y el empleo de entornos Entornos Tridimensionales (3Ds) entraron en juego en varias asignaturas. Realmente disfruté al trabajar con estas tecnologías y esto me llevó a mi siguiente paso.

Cuando en mi mente me planteé el desarrollo de un proyecto de final de grado como este que está leyendo, jamás concebí un escenario que no concediera una mejora a la hora de realizar una determinada acción, en concreto mejorar la calidad de vida de personas con algún tipo de discapacidad. Por ello cuando mi tutor me comentó el proyecto que llevaban entre manos, sin dudarlo un instante supe que ese era el camino que debía seguir.

Esto me lleva a un punto nuevo, lleno de retos y problemas por solventar, trabajo en equipo y horas y horas de insaciable investigación para hacer la vida de alguien un poco mejor. Ayudar en tareas cotidianas del hogar, a personas que siguen luchando, y por supuesto, nosotros luchamos junto a ellas. Si con este planteamiento no ha despertado vuestro lado más curioso, quizás los siguientes capítulos llenos de información, gráficas y planteamientos no son para usted.

## 1.3. Trabajos relacionados

Comenzaré hablando sobre uno de los problemas más comunes a la hora de trabajar con redes neuronales de cualquier tipo, los datos. Este es un problema que va de la mano junto con los algoritmos de aprendizaje automático, cuanto mejores sean mis datos, mejores resultados es probable que obtenga. Pero ojo, se precisa una de gran calidad en los datos, también es necesario una gran volumen de estos, aunque no siempre es posible. En nuestro caso, no es viable obtener estos datos de entornos reales como casas, residencias, hospitales... puesto que para redes con modelos muy complejos es necesario una gran cantidad de datos para evitar el overfitting [1], y resultaría muy laborioso obtener estos datos de entornos reales. Pero si echamos la vista hacia temas de generación sintética de datos, es posible encontrar información que arroje un poco de luz a este punto.

En esta sección voy a tratar aspectos, desarrollos y publicaciones más destacables en la generación de datos sintéticos en entornos controlados hasta la fecha. Aunque las técnicas aplicadas difieren unas de otras, la gran mayoría coinciden en lo mismo [1]-[3], la necesidad de dotar al robot del conocimiento necesario para completar las tareas en escenarios desconocidos. Para ello se adoptan distintas estrategias, y como resultado obtienen el llamado Conjunto del conocimiento base para un ámbito en concretos (KBs) que recoge todos los datos procesados y tratados que conforman los casos de aprendizaje para el robot. Para crear el ansiado dataset sintético, será necesario este KBs para posteriormente generar el mayor número de casos posibles con los que poder trabajar. Como parte final solo resta desarrollar un modelo especializado para el tratamiento de este tipo de acciones, y para ello también se presentan múltiples opciones a analizar.

### 1.3.1. Entorno virtual interactivo

Para la generación del conjunto de datos, numerosos casos se apoyan en entornos 3Ds para facilitar el manejo de diferentes tipos de acciones humanas al trabajar con entornos virtuales. Esta práctica la realizan desarrollos que experimentan con robots y su interacción con el entorno [4]-[6].

En el caso de TORCS [4] han creado un entorno completamente funcional centrado en la simulación automovilística. En este proyecto no priman los gráficos en sus escenarios, si no: la interacción de los coches con la pista, la actuación de los neumáticos o suspensión con la climatología, el estado del circuito... Gracias a esta herramienta se pueden entrenar algoritmos de aprendizaje automático con la capacidad de adquirir una autonomía suficiente para rivalizar con un humano. Lo que podemos destacar es

[1] http://www.aprendemachinelearning.com/que-es-overfitting-y-underfitting-y-como-solucionarlo/

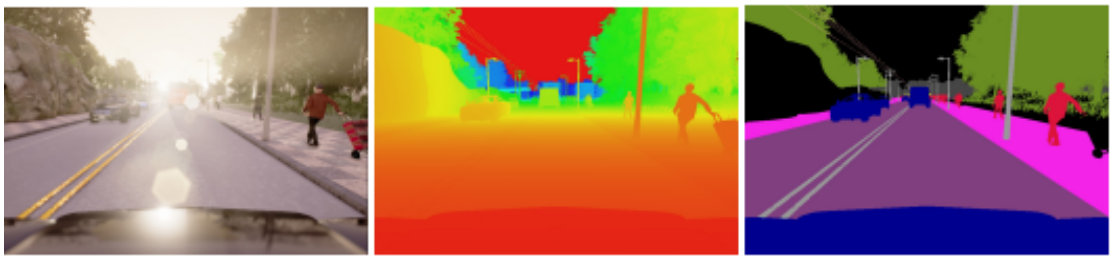


**Figura 1.1:** En la izquierda se aprecia la imagen en RGB, en el centro la profundidad y en la derecha la segmentación. Figura extraída de [6].

esta relación, en otros trabajos también se prima esta interacción [5], [6] más allá de los gráficos empleados.

Este último trabajo nombrado [6] ha sido desarrollado en Unreal Engine 4 [7], y todas las texturas se han mantenido con el menor número de polígonos posibles, pero manteniendo una calidad visual superior al anterior artículo. Al igual que TORCS [4], se trata de un entorno centrado en la conducción, pero esto no es relevante para mi estudio, puesto que no voy me centro en mecánicas de movimiento, sino de segmentación de acciones. Lo que si es relevante sobre todo son los comportamientos en entornos habitables dotados a los robots, y que posteriormente pueden ser analizados, y todos los sensores que incorporaron a estos para obtener datos detallados del entorno. Incluyen cámaras con las que extraen las imágenes reales en Modelo aditivo de colores Rojo, Verde, Azuls (RGBs), imágenes de profundidad y de separación de bordes, estás últimas separan los objetos identificados en la escena. Una pequeña representación de estas imágenes obtenidas se puede ver en la *Figura* 1.1.

### 1.3.2. Dotar al robot de conocimiento

Cuando pensamos en un robot, se nos vienen a la mente multitud de imágenes de todo tipo, desde los famosos *Roomba* que nos limpian y friegan el suelo, hasta coches que son capaces de conducir de forma autónoma, con mayor o menor precisión. Si intentamos indagar un poco sobre el funcionamiento de estos sistemas, rápidamente acabamos en artículos que implementan algoritmos de inteligencia artificial, como pueden ser métodos basados en Machine Learning. Pero ¿Tan mágicos son estos algoritmos? ¿Funcionan sin más y nos olvidamos del resto tratándolo a modo de caja negra? ¿Tan complicado es esto para poder entenderlo de forma sencilla?

A mi parecer, todas estas preguntas tienen por respuesta un **NO** rotundo. Pero también es necesario, aunque suene muy insólito, pensar que el robot es como un bebé. Un bebé nace, y en este momento va aprendiendo como funciona el entorno que le rodea, aprende que cosas puede hacer y que no, aprende como interaccionar con otras personas e incluso, con el tiempo adecuado, a realizar acciones más complejas. Pues bien, en entornos 3Ds como los explicados en la sección anterior, un robot puede aprender como funciona el entorno, sus limitaciones, como interaccionar con otros objetos o robots, e incluso a realizar ciertas tareas más complejas. Pero claro, aquí se nos plantea una gran diferencia entre el bebé y el robot (a parte de que un robot no está vivo, como si lo está el bebé), un gran conjunto de conocimientos que el bebé va desarrollando por métodos didácticos que se escapan al ámbito de esta investigación. Pero claro ¿Como puede el

robot tener estos conocimientos y experiencias?.Desde el punto de vista adecuado, esto es relativamente sencillo, y es preciso introducir el concepto de KBs.

Para comenzar es necesario disponer de una KBs. Para ello se puede optar por solicitar a una parte del personal que introduzcan descripciones de acciones cotidianas en los hogares [1]. Estas descripciones van seguidas de una serie de acciones atómicas que se introdujeron junto con la definición de la actividad y que fueron comprobadas y verificadas. La explicación de esto es bien sencilla, es para dotar al sistema de sentido común en las acciones realizadas. Por ejemplo, una acción frecuente como "Servir agua en una taza" implica el conocimiento previo para saber que tienes que coger la taza, luego el agua, abrir la botella (en caso de ser una botella) y verter el líquido en la taza. Todo esto es fruto de una sucesión de secuencias atómicas para cada actividad. Por ello, desarrollaron un software propio basado en Scratch [2] para etiquetar las acciones con ayuda de un entorno 3Ds y a su vez almacenar las grabaciones del entorno mientras se ejecutaban diferentes actividades. Para los entornos se han recreado varias estancias que permiten a los personajes interactuar con el entorno virtual de manera similar al artículo [5]. Para más información sobre esta generación de datos, se puede consultar la siguiente dirección `http://virtual-home.org/`.

Otros [3] optan por generar estos conjuntos a través del etiquetado de actividades grabadas en vídeo. Por otro lado, en el caso de [2] se optó por acceder directamente a recursos en linea para obtener este conjunto de datos, el cuál estaba formado por 53.000 instrucciones de acciones relevantes en entornos cotidianos. A pesar de la gran cantidad de datos disponibles, esta propuesta tenía el mismo problema que el anterior artículo, estas definiciones simplemente trataban de forma general las acciones pero sin entrar en detalle sobre que hacer en cada uno de los instantes, lo cuál es un gran problema de ambigüedad, ya que puede ocasionar situaciones de gran imprecisión para el robot.

El paso posterior a la obtención de las descripciones de las acciones, como se ha comentado anteriormente, es la eliminación de cualquier tipo de ambigüedad que puedan aparecer en las definiciones. Tomando como ejemplo la acción "Dar la vuelta", que se caracteriza por su gran imprecisión, aunque nosotros seamos capaces por el contexto de darle el significado correcto y actuar en consecuencia. Esto es lo que se plantea y se intenta solucionar para que los robots a su vez, también sean capaces de actuar correctamente cruzando esto con su conocimiento previo de la escena. Junto a esto también deben saber cuando acaba la acción, ya que es algo muy evidente para nosotros, pero que también debería serlo para las máquinas, o al menos así lo plantean. Para resolver este problema [2] se apoya en el Probabilistic Robot Action Coress (PRACs) para representar una distribución de probabilidad conjunta sobre la acción de tal manera que la acción de entrada pueda servir para obtener la información suficiente sin ambigüedad para completar la acción Figura 1.2.

## 1.4. Datasets más relevantes

Los datasets actuales son muy variados. Para el propósito de este proyecto se pueden encontrar desde conjuntos de datos con imágenes del interior de casas, hasta datasets con escenarios en ambientes al aire libre, como pueden ser entornos urbanos. Para

[2] `https://scratch.mit.edu/`

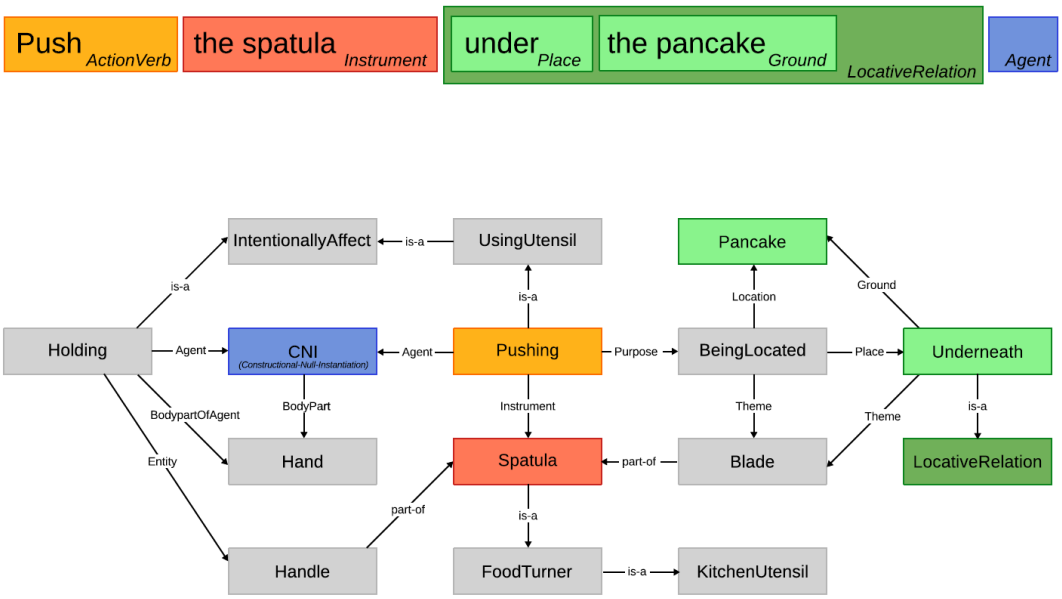



**Figura 1.2:** Ejemplo de actuación de Robot Action Core para la instrucción Çolocar la espátula debajo del crepe". Se pueden apreciar las relaciones que se han generado a partir de un conocimiento previo de los objetos de la escena. Figura extraida de [2].

una mayor percepción de estos datos comentados, vamos a ver en detalle algunos de los más importantes a tener en cuenta, si de generar datos consta nuestro objetivo.

### 1.4.1. Virtual Home

Este es un dataset [1] ya comentado en secciones anteriores (Sección 1.3.1). Pero no se trata en concreto de un dataset, simplemente se hace referencia a este conjunto de datos. Realmente este es un proyecto basado en la simulación de actividades en entornos cotidianos por medio de programas informáticos. Lo realmente curiosos de los datos que manejan, es la cantidad de acciones incluidas en estos y las relaciones que tienen unas con otras. Puesto que esta diseñado de tal forma que mediante la asignación de diferentes pasos, se permite recomponer una actividad donde estén involucradas estas acciones. Esto se realiza empleando un entorno y lenguaje basado en Scratch, dotado de gran facilidad para componer código o módulos y desarrollar los conjuntos de actividades deseado. Esta secuencia de pasos indicada, se puede apreciar con mayor claridad en la Figura 1.3.

$$
\begin{aligned}
\text{step}_1 &= [\text{Walk}]\ \langle \text{TELEVISION} \rangle (1) \\
\text{step}_2 &= [\text{SwitchOn}]\ \langle \text{TELEVISION} \rangle (1) \\
\text{step}_3 &= [\text{Walk}]\ \langle \text{SOFA} \rangle (1) \\
\text{step}_4 &= [\text{Sit}]\ \langle \text{SOFA} \rangle (1) \\
\text{step}_5 &= [\text{Watch}]\ \langle \text{TELEVISION} \rangle (1)
\end{aligned}
$$

**Figura 1.3:** Pasos establecidos en la acción, junto con las opciones asociadas a estos. Se aprecian los objetos con los que interactua y la cantidad de los mismos.

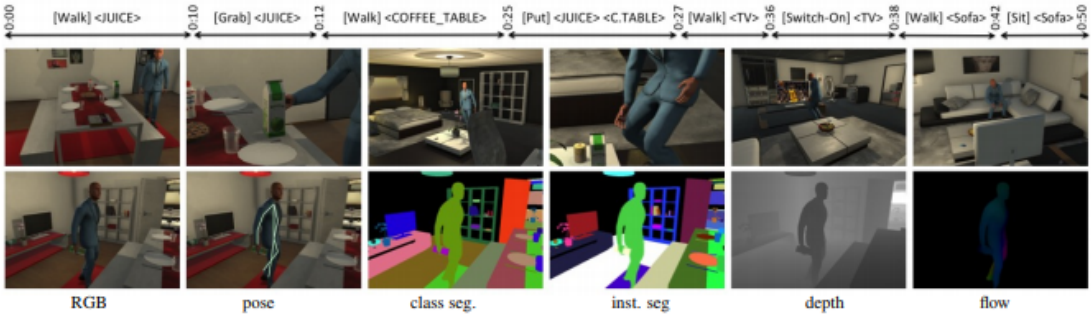



**Figura 1.4:** Se aprecian diferentes acciones desempeñadas por el actor en la fila superior. En la fila inferior, se pueden ver los tipos de filtros que emplean las cámaras para extraer las representaciones y características de la escena.

Cubre un total de 75 acciones atómicas, las cuales tienen la posibilidad de interaccionar con diferentes tipos de objetos en la escena. Este tipo de características resulta de gran importancia, ya que gran cantidad de las tareas del hogar se realizan empleando cualquier tipo de objeto (Figura 1.3). Además, cabe destacar la gran cantidad de escenarios empleados para la construcción del dataset, en concreto ocho escenas diferentes. Este *cocktail* da como resultado un conjunto de datos enormemente variado, capaz de construir gran cantidad de acciones cotidianas. Para la generación de estas secuencias, por medio únicamente de una descripción detallada de la acción deseada, se extrajeron diferentes tipos datos en la escena (Figura 1.4). Esta aproximación facilita gran información a sistemas de inteligencia artificial que permiten analizar secuencias de imágenes, dicho propósito es el objetivo de generar un dataset sintético por medio de Unreal Engine 4.

### 1.4.2. SYNTHIA

Un gran conjunto de imágenes sintéticas para segmentación de escenas urbanas, este es el título que establecen desde la Universidad de Barcelona y la Universidad de Viena [8] para este dataset. Ha sido generado con el propósito de ser empleado para segmentar las 13 acciones diferentes que incluyen en su contenido. Estas actividades son diversas, y abarcan desde caminar por la acera hasta escalada o conducción de vehículos. Para visualizar mejor estas actividades, se puede observar la Figura 1.5 Todos estos datos se adquieren a su vez, desde diferentes puntos de vista, empleando diversas cámaras desplegadas en el entorno. Para aumentar la variedad de escenarios, juegan con las estaciones del año, cambiando el aspecto del entorno en función del que se desee.

Para capturar estos datos, emplean arrays de cámaras situados por la escena que capturan información variada de esta. A su vez, rellenan el entorno con objetos realistas como pueden ser: coches, ciclistas... Estos coches también están dotados de cámaras que permiten obtener otro punto de vista de las escenas. Estas cámaras tienen una libertad de movimiento de hasta 100 grados y pueden grabar hasta 800 metros de distancia, suficiente para obtener información útil de las diferentes acciones. Además, el array de cámaras colocado en la escena, se mueve aleatoriamente a medida que la animación se ejecuta.

**Figura 1.5:** Conjunto de acciones capturadas en el dataset.

### 1.4.3. PHAV

En este caso, nos encontramos de nuevo con un dataset establecido en un escenario urbano y en entornos cerrados [9]. Estos lugares son variados, ya que podemos encontrar campos de fútbol, ciudades o pueblos, en los que los actores se mueven a lo largo de la escena. Pero no solo consta de personajes principales, también se incluyen actores secundarios en segundo plano de la escena, que pueden llegar a interactuar y completar la actividad. Incluir diversos objetos en escena aumenta esta variedad, y además dota de información relevante para mejorar la comprensión de la zona [10], [11]. Para aumentar la diversidad en los datos, se emplean variaciones físicas en los huesos del esqueleto, siempre dentro de unos límites plausibles. Ciertamente, esta técnica es de gran utilidad, puesto que abarca la opción de representar movimientos con esqueletos que presenten algún tipo de discapacidad física. Aunque estas perturbaciones físicas no solo se aplican en los esqueletos o el movimiento del personaje, también a diferentes objetos en la escena. Esto puede provocar situaciones que causan errores en el renderizado, como es el caso de atravesar paredes.

Por otro lado, al igual que ocurría con el dataset anterior, es posible alterar las condiciones climáticas del escenario, como pueden ser: lluvia, nieve, niebla... En la fase de grabación, apuestan por un plano diferente. Sitúan una cámara en la parte trasera del actor, permitiendo un grado de movimiento a esta para realizar diferentes movimientos a la hora de grabar la acción (no se trata de cámaras con posiciones estáticas durante la reproducción), estas incluyen tareas como: mover objetos, pasear con otros actores o ser golpeado por un coche (ver Figura 1.6). Estas actividades son muy diversas, entre todas estas incluyen 2605 secuencias diferentes de 144 tipos divididas en 6 categorías, a su vez estas están etiquetadas con un pequeño texto, que servirá para entrenar un modelo de aprendizaje supervisado.

### 1.4.4. Inteligencia Artificial para segmentar acciones de vídeo

Una construcción elaborada y sólida de nuestro dataset, va a ser la base de nuestro desarrollo, pero con el propósito de emplearlo para segmentar acciones en vídeo. Esta segmentación se ha empleado mediante técnicas de inteligencia artificial a lo largo de diferentes proyectos. Incluso en los conjuntos de datos vistos en la sección anterior, se

**Figura 1.6:** Algunos ejemplos de actividades que contiene el conjunto de datos: empujar o golpear (en la parte superior) y recibir un impacto o pasear acompañado (en la parte inferior).

pueden encontrar estas técnicas aplicadas, o bien para identificar acciones, o para detectar objetos.

Como punto de partida, se pueden apreciar proyectos que emplean árboles de decisión para la segmentación de estas acciones [3], [12]. Estas técnicas se emplean para separar semánticamente las subacciones involucradas a partir de una pequeña descripción de la actividad, generando para ello Gráficos de visualización semánticas (SVGs) [3]. Esto permite estructurar estas subacciones y poder diferenciar cuales están involucradas en diferentes tareas. Incluso, se puede descubrir el espacio semántico no visible a simple vista en el que se desarrolla la escena. Realmente, el propósito del uso de estos árboles, fue dotar a un robot de la capacidad para manipular objetos. Por otro lado, estos árboles son empleados para detectar la posición de un esqueleto y saber las zonas de unión de los huesos [12]. En concreto, se han empleado tanto árboles de decisión como random forest, debido que resultaron ser muy rápidos y efectivos.

En el lado de técnicas como es el caso de las redes neuronales, se pueden encontrar multitud de desarrollos interesantes que analizan o segmentan estas acciones [6], [8], [9], [13]. Incluso en este punto, se pueden encontrar diferentes apuestas en lo que respecta al tipo de entrenamiento. Por un lado podemos apreciar el uso de técnicas de entrenamiento por refuerzo [6], [13], para realizar análisis de la escena en cuestión o controlar de forma autónoma un vehículo respectivamente. Mucho más común es encontrarnos redes neuronales que aplican un modelo de aprendizaje supervisado. En este caso encontramos proyectos aplicados al reconocimiento de actividades en vídeo [9] y segmentación de acciones en entornos urbanos [8]. Empleando para ello las imágenes RGBs obtenidas en el dataset y las ground truth generadas en cada una de ellas. En la Figura 1.7 se puede apreciar la comparación de estas salidas, en los conjuntos de entrenamientos con diferentes entradas.

Lo cierto es que el uso de estas técnicas permite obtener una segmentación de cada uno de los componentes que aparecen en la escena, permitiendo comprender cual es la acción realizada y los actores u objetos que forman parte en ella. Ya sea a través de técnicas basadas en deep learning o más tradicionales como el uso de árboles, esta

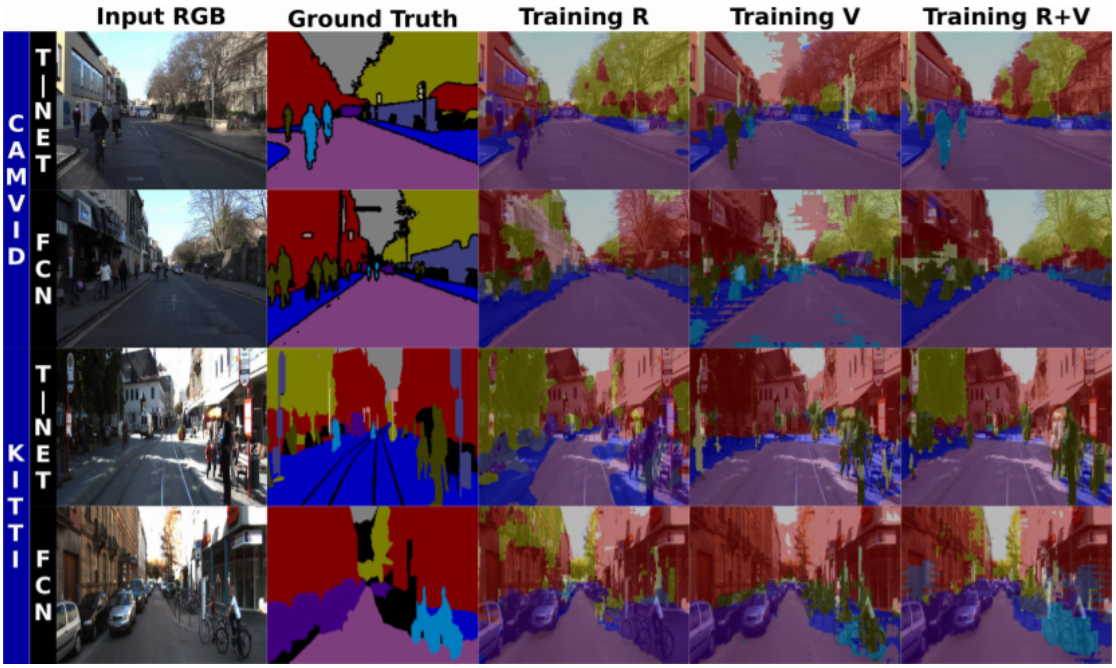



**Figura 1.7:** Resultados para diferentes conjuntos de datos y arquitecturas de pruebas (T-Net y FCN). La primera columna muestra las pruebas con imágenes RGB, la segunda columna se trata de las ground thruth. El entrenamiento del conjunto R es el resultado del entrenamiento con el conjunto de datos real. El Entrenamiento del conjunto V es el resultado del entrenamiento con el dataset SYNTHIA [8]. El entrenamiento del conjunto R+V es el resultado del entrenamiento con la colección real y SYNTHIA.

claro que emplear métodos de aprendizaje automático es necesario si se desea dotar a un robot de la percepción del entorno y qué ocurre a cada momento a su alrededor.

## 1.5. Planificación

Este ha sido un proyecto elaborado en 197 días repartido en diferentes etapas, las cuales serán descritas a continuación. El 10 de septiembre de 2018 se comenzó con la primera fase del desarrollo, que se prolongó hasta el 31 de mayo de 2019, cuya fecha corresponde a la entrega del trabajo de final de grado.

### 1.5.1. Planificación general

En la Figura 1.8 se diferencian las diferentes partes que se abordaron en el trabajo, desde la formación hasta el desarrollo final.

### 1.5.2. Formación

El punto de partida para este proyecto, resulta ser una fase de aprendizaje (Figura 1.9). En concreto se aprendieron conceptos relacionados con Unreal Engine 4 y con técnicas de aprendizaje automático, puesto que la construcción del dataset tiene como objetivo ser utilizado para este propósito.

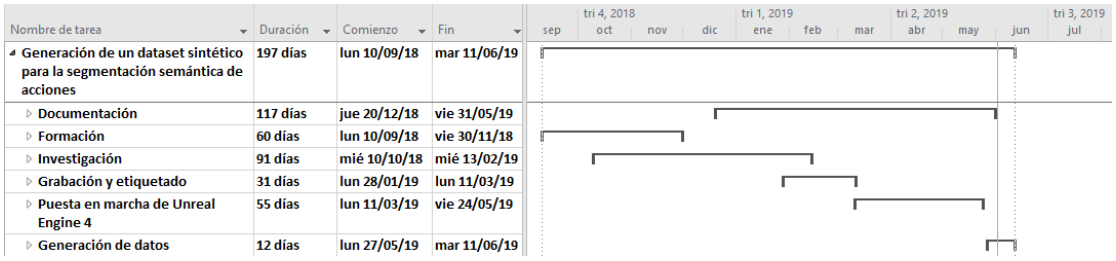



**Figura 1.8:** Planificación general del proyecto.

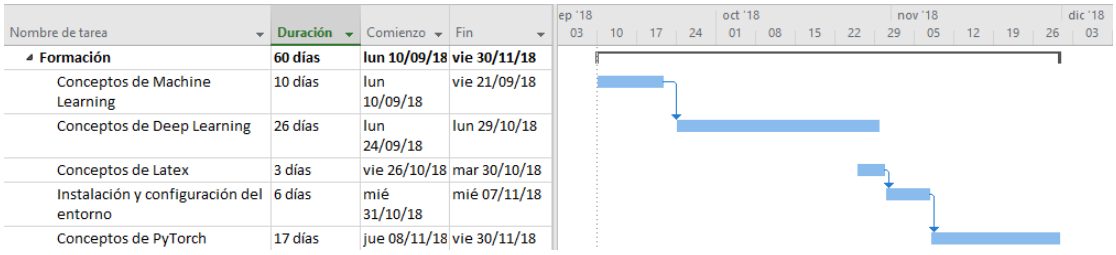



**Figura 1.9:** Planificación de la formación del proyecto.

### 1.5.3. Investigación

El estado del arte actual se llevo acabo a lo largo de una gran cantidad de tiempo. Se investigó sobre temas concretos de creación y uso de estos conjuntos de datos. Además resultó de gran interés aprender un poco más acerca de como nos movemos los humanos, y de que forma replicarlo y analizar estas acciones.

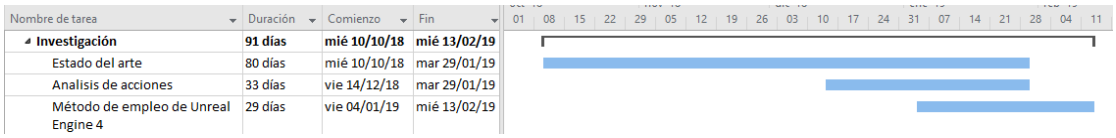


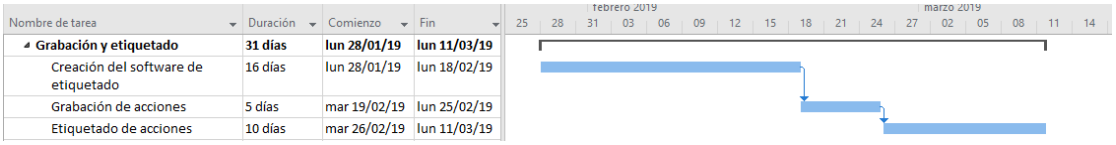

**Figura 1.10:** Planificación de la investigación del proyecto.

### 1.5.4. Grabación y etiquetado

Fue preciso poner en práctica la información analizada durante el estado del arte, puesto que era necesario emplear el traje del Axis Neuron (2.1.2) para replicar este conjunto de acciones de la forma más humana posible. Posteriormente se empleó el script de etiquetado para establecer las acciones y subacciones correspondientes a cada frame.

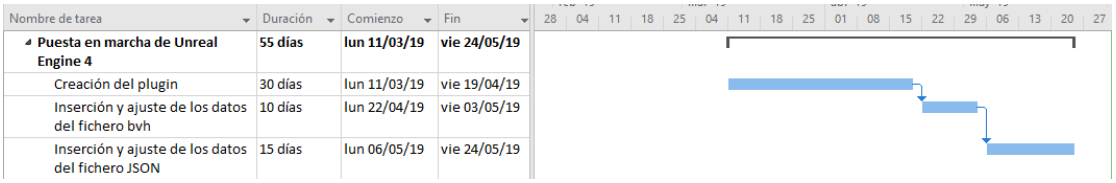
### 1.5.5. Puesta en marcha de Unreal Engine 4

Unreal Engine 4 se utilizo durante un gran periodo debido a la necesidad de incluir la lógica necesaria para controlar al esqueleto por medio de los ficheros BVH. A continuación, se tubo que ajustar UnrealROX para que ejecutara cada frame de esta animación, ya que como se comenta en la Sección 4.2, el objeto llamado Tracker acctua como orquestador de la reproducción de las diferentes acciones.



| Nombre de tarea | Duración | Comienzo | Fin |
|---|---|---|---|
| **Grabación y etiquetado** | **31 días** | **lun 28/01/19** | **lun 11/03/19** |
| Creación del software de etiquetado | 16 días | lun 28/01/19 | lun 18/02/19 |
| Grabación de acciones | 5 días | mar 19/02/19 | lun 25/02/19 |
| Etiquetado de acciones | 10 días | mar 26/02/19 | lun 11/03/19 |

**Figura 1.11:** Planificación de la fase de grabación y etiquetado del proyecto.

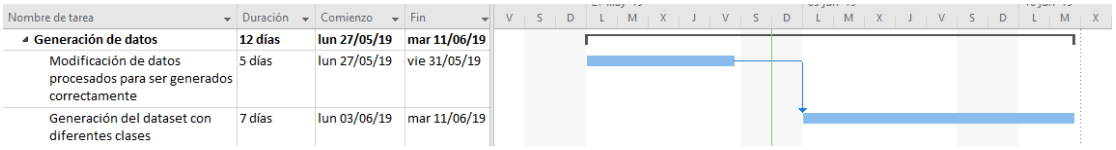


| Nombre de tarea | Duración | Comienzo | Fin |
|---|---|---|---|
| **Puesta en marcha de Unreal Engine 4** | **55 días** | **lun 11/03/19** | **vie 24/05/19** |
| Creación del plugin | 30 días | lun 11/03/19 | vie 19/04/19 |
| Inserción y ajuste de los datos del fichero bvh | 10 días | lun 22/04/19 | vie 03/05/19 |
| Inserción y ajuste de los datos del fichero JSON | 15 días | lun 06/05/19 | vie 24/05/19 |

**Figura 1.12:** Planificación de las fases que emplearon Unreal Engine 4 para su desarrollo.

### 1.5.6. Generación de datos

La generación de datos fue el periodo en el cual ya se disponía de las herramientas necesarias y todos los componentes listos. Por tanto, corresponde con el espacio de tiempo necesario para generar un dataset completo, incluyendo en este tiempo, errores a solventar.



| Nombre de tarea | Duración | Comienzo | Fin |
|---|---|---|---|
| **Generación de datos** | **12 días** | **lun 27/05/19** | **mar 11/06/19** |
| Modificación de datos procesados para ser generados correctamente | 5 días | lun 27/05/19 | vie 31/05/19 |
| Generación del dataset con diferentes clases | 7 días | lun 03/06/19 | mar 11/06/19 |

**Figura 1.13:** Planificación de la fase de generación de datos del proyecto.

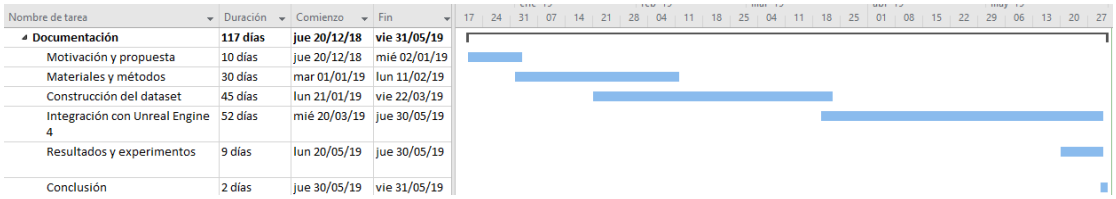
### 1.5.7. Documentación

Durante todo el periodo que duró el proyecto, fue necesario ir construyendo en parte la documentación. Esto se realizó en paralelo al desarrollo del resto del trabajo.

## 1.6. Resumen

En primer lugar, se ha comprobado la gran necesidad que se tiene de generar una gran cantidad de datos cuando de inteligencia artificial se trata. Estos datos, bien sea a mano o de forma automática, son realmente muy laboriosos de generar, pero resultan el pilar fundamental de un buen desarrollo. Para obtener un buen resultado, es imprescindible generar un dataset abundante, a la vez que variado en clases y escenarios. Por tanto, emplear técnicas como aumentar cámaras, dotar de movimiento a estas cámaras, incluir elementos en la escena, interactuar con estos objetos o incluso modificar la estación del año son acciones que te permiten mejorar los datos generados.

A su vez, emplear algoritmos de inteligencia artificial capaces de trabajar con estas imágenes (RGB, profundidad, normales...) y aprender a segmentar estas acciones es



| Nombre de tarea | Duración | Comienzo | Fin |
|---|---|---|---|
| **Documentación** | **117 días** | **jue 20/12/18** | **vie 31/05/19** |
| Motivación y propuesta | 10 días | jue 20/12/18 | mié 02/01/19 |
| Materiales y métodos | 30 días | mar 01/01/19 | lun 11/02/19 |
| Construcción del dataset | 45 días | lun 21/01/19 | vie 22/03/19 |
| Integración con Unreal Engine 4 | 52 días | mié 20/03/19 | jue 30/05/19 |
| Resultados y experimentos | 9 días | lun 20/05/19 | jue 30/05/19 |
| Conclusión | 2 días | jue 30/05/19 | vie 31/05/19 |



**Figura 1.14:** Planificación general de la documentación del proyecto.

una ardua tarea. Por ello, desarrollos como los vistos en este capítulo, determinan la favorable actuación de los robots entrenados con estos sistemas. Diferenciando entre ellos, algoritmos específicos para cada tarea, puesto que no todas emplean los mismos tipos de datos, y por ende, tampoco serán iguales estos desarrollos.

# Capítulo 2

# Materiales y métodos

*Los apartados correspondientes a este segundo capítulo, recogen el conjunto de materiales (software y hardware) y métodos llevados a cabo durante el proceso de investigación y desarrollo. Siendo más específico tenemos secciones con un análisis de los materiales empleados 2.1, 2.2 y 2.3.*

## 2.1. Hardware

Los procesos de generación de datasets sintéticos y de aprendizaje profundo, son técnicas actuales que precisan de componentes de última generación para reducir tiempos de procesado y aumentar la calidad. Gran parte de estos componentes no hubiera sido posible emplearlos sin la ayuda del Departamento de Tecnología Informática y Computación de la Universidad de Alicante (DTIC), entre ellos encontramos los siguientes.

### 2.1.1. Clark

Clarke es un superordenador que pertenece al DTIC y esta alojado en laboratorio de I+D 2 en la segunda planta de la Politécnica III. Este es un ordenador equipado con GPUs en concreto para el entrenamiento de redes neuronales de aprendizaje profundo. Sus características hardware son las siguientes:

- CPU

  En el aspecto de la CPU presenta i7-6800K con estas funcionalidades:

  - 6 cores a 3,40 GHz, Max Turbo 3,80 GHz.
  - 15 MB Smart cache.
  - 64 bit, SSE 4.2,AVX2.
  - 14 nm , Max TPD 140 w.
  - Soporte Hyperthreading.

  Además de las características CPU, Clarke presenta estas especificaciones hardware:

  - Front Bus 1333-2133.
  - Memoria 16 GB DDR4-2666.
  - 2 discos SATA 7,2 k, 3TB (RAID1).
  - 1 disco SATA SSD, 500GB.

Pero las partes hardware más importantes son las tres tarjetas gráficas que dispone. Dos de ellas son las que se utilizan para trabajar en [problemas/proyectos] de inteligencia artificial y operaciones de cálculo y la tercera está dedicada para salida de vídeo en tareas de mantenimiento. Las especificaciones son las siguientes:

- NVIDIA Quadro 2000


**Figura 2.1:** Tarjeta NVIDIA Quadro 2000

  - Arquitectura: Fermi
  - Memoria: 1024 MB DDR5
  - Núcleos: 192 CUDA Cores
  - Interfaz de memoria: 128-bits
  - Ancho de banda de la memoria: 41,6 GB/s

- NVIDIA Tesla K40c


**Figura 2.2:** Tarjeta NVIDIA Tesla K40c

  - Arquitectura: Kepler

- Memoria: 12 GB GDDR5
- Núcleos: 2880 CUDA Cores
- Interfaz de memoria: 384-bits
- Ancho de banda de la memoria: 288 GB/s

La primera que se muestra en el listado es la tarjeta orientada a salida de vídeo para operaciones de depuración del ordenador o fallos puntuales. Aparte de la tarjeta para visualizar, Clarke dispone de dos K40c, estas tarjetas están orientadas para centros de datos y superordenadores y por ello no disponen de salida gráfica, la potencia de su GPU esta orientada principalmente a problemas que puedan ser paralelizados. Esta tarjetas presentan una potencia de 4.29 TFLOPS. Para nuestro caso, esta cantidad de operaciones por segundo nos va a permitir entrenar los modelos mucho más rápido que empleando la CPU o tarjetas gráficas convencionales.

### 2.1.2. Perception Neuron

Esta innovadora herramienta se trata de un traje compuesto por múltiples neuronas interconectadas entre sí, que juntas forman un esqueleto muy similar al humano. Entre sus características más importantes se puede destacar la capacidad para capturar el movimiento del actor de forma muy adaptable al medio y muy versátil al tratarse de un traje que permite la movilidad de sus neuronas. El diseño del traje comentado puede apreciarse en la Figura 2.3.

Se dispone de diferentes versiones, que varían el número de neuronas disponible, en el caso de este proyecto se ha empleado un total de 32 sensores. Estos van desde los pies y tobillos hasta el cuello y cabeza.

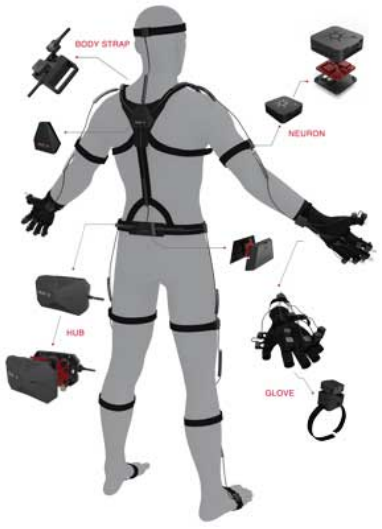



**Figura 2.3:** Esqueleto del Perception Neuron sobre un cuerpo humano

Los dos componentes principales del esqueleto son:

- Sensores o neuronas.

**Figura 2.4:** Neuronas que componen el esqueleto

- Tamaño: 12.5mm X 13.1mm X 4.3mm.
- Rango dinámico: 360 grados.
- Rango del acelerómetro: ś 16g.
- Rango del giroscopio: ś 2000 dps.
- Resolución: 0.02 grados.
- Tasa de salida máxima: 60fps w / 32 Neuronas, 120fps w / 18.
- Batería USB externa: 5V / 2AMP.

- Neuron HUB.


**Figura 2.5:** Neuronas que componen el esqueleto

- Tamaño: 59 mm x 41 mm x 23 mm.
- Mínimos sensores conectados: 3.
- Máximos sensores conectados: 32.
- Salida: USB 2.0, Wi-Fi o grabación en tarjeta Micro-SD incorporada.
- Velocidad de salida máxima: 60 fps con 32 neuronas, 120 fps con 18 neuronas.
- Alimentación: batería externa USB.

Como se ha comentado, esto me ha permitido grabar acciones yo mismo con mayor o menor grado de error, pero suficientemente preciso para permitir que una red neuronal aprenda con estas grabaciones. Para obtener más información a cerca de este dispositivo, es interesante consultar la siguiente referencia [15].

## 2.2. Software

En cuanto al software empleado, ha dependido del estado del proyecto, ya que para cada una de las tareas era necesaria una tecnología diferente. En este punto he visto por conveniente diferenciar entre el software como tal empleado, y los frameworks como apartado adicional, los cuales también han tenido gran peso en el desarrollo. Estos últimos serán explicados en el capítulo siguiente.

### 2.2.1. Python

Python se trata de un lenguaje de programación multiparadigma, esto nos permite trabajar con programación imperativa, funcional u orientada a objetos, además que permite la ejecución en multiples sistemas operativos. Estas características hacen de python una pieza fundamental en el desarrollo de este proyecto.

El uso que se le da a este lenguaje se divide en dos partes:

- Script de etiquetado: Se ha empleado para generar un script que permite el etiquetado de las acciones grabadas mediante el traje Axis Neuron.
- Red neuronal: Es el lenguaje por excelencia en la creación de modelos de aprendizaje automático. Por tanto, python también lo he empleado para desarrollar la red neuronal capaz de trabajar con el dataset creado.

Para obtener más información a cerca de esta tecnología, es interesante consultar la siguiente referencia [16].

### 2.2.2. Axis Neuron

Esta herramienta nos permite comunicar nuestro ordenador con el esqueleto que he empleado para las grabaciones (2.3). Esta comunicación nos permite realizarla por medio de USB o WIFI, en mi caso la opción escogida ha sido la primera. También podemos encontrar dos versiones de este programa, la versión demo y pro. La primera es gratuita y nos permite trabajar con las funcionalidades principales del traje, la segunda nos aporta todas las características y opciones de grabación y exportación.

Una vez dentro del entorno de grabación, podremos ver como se divide en tres áreas principales, reflejadas en la Figura 2.6.

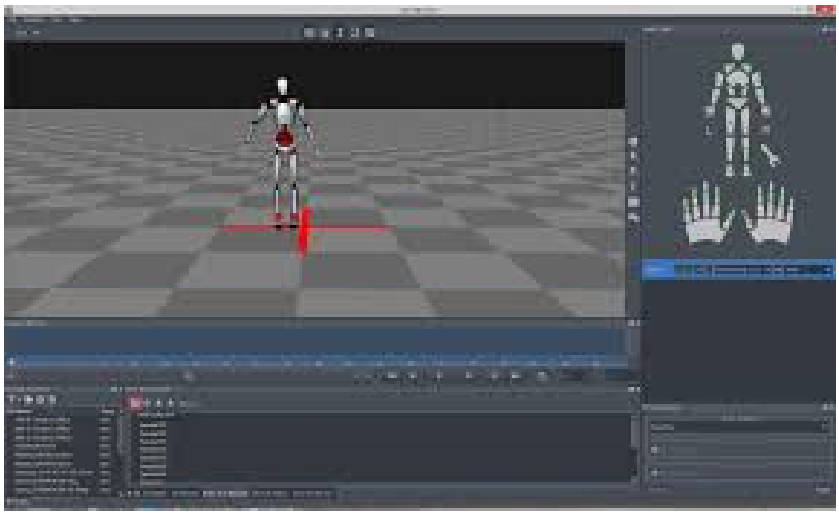


**Figura 2.6:** Entorno de grabación del Axis Neuron

- Vista previa del esqueleto: Nos permite ver en tiempo real el movimiento que está realizando el traje mientras está conectado al ordenador. En el también se puede reproducir una grabación previamente realizada.

- Huesos conectados y características del esqueleto: Nos permite ver las neuronas conectadas en cada momento y ajustar las características de estas.
- Explorador de archivos y características de grabación: Nos permite seleccionar los archivos que deseamos cargar para y reproducir ajustando sus características.

Para obtener más información a cerca de esta tecnología, es interesante consultar la siguiente referencia [17].

### 2.2.3. Blender

Blender se trata de una herramienta multiplataforma, con especial dedicación al modelado 3D, iluminación, renderizado, animación y creación de gráficos tridimensionales. De forma adicional, nos permite emplear técnicas procesales de nodos, edición de video, escultura y pintura digital. Gracias al uso de su motor interno de juegos, esta herramienta nos permite crear videojuegos, aunque esta no es la funcionalidad que le he dado en este proyecto.

Al trabajar con esqueletos, mallas y animaciones, el manejo de blender ha resultado esencial. Ha permitido observar, modificar y generar las estructuras de huesos que componen las diferentes mallas y animaciones empleadas. Además me permitió reproducir las animaciones *frame a frame*, dicha práctica resulta vital para etiquetar las animaciones, valga la redundancia,*frame a frame*. Por último, Axis Neuron (2.6), no permite una exportación de Biovision Hierarchy (BVH) a Filmbox (FBX) de forma correcta, lo cual también añade un punto al marcador por parte de blender, cuya transformación ha sido de gran utilidad en todo el proceso.

Para obtener más información a cerca de esta tecnología, es interesante consultar la siguiente referencia [18].

### 2.2.4. Unreal Engine

Unreal Engine se trata de un motor de juego para Ordenador Personal (PC) y consolas creado por la compañía Epic Games. Mediante el uso de C++, el motor gráfico presenta un alto grado de portabilidad y es una herramienta empleada actualmente por muchos desarrolladores de juegos. Otro factor clave para esta herramienta, es su uso gratuito y la gran comunidad que da soporte al motor. Esta herramienta está en continua evolución y hasta la fecha la versión más actualizada es la 4.22, la cuál data de abril de 2019. A pesar de esto, para el desarrollo llevado a cabo, se esta trabajando sobre la versión 4.18 por restricciones con el software/pluging/unrealrox...

Unreal Engine 4 ha aportado una gran cantidad de facilidades para trabajar con los esqueletos, mallas y animaciones. Aunque no se ha estado desarrollando un videojuego, emplear las herramientas dedicadas a ello permite trabajar con más soltura. Por ende, ha sido un factor clave para cargar los ficheros BVH junto con las mallas del actor y generar datos a partir de toda la información grabada previamente con el traje.

Para obtener más información a cerca de esta tecnología, es interesante consultar la siguiente referencia [7].

### 2.2.5. Google Colab

La aportación de Google a la tecnología actual es sobresaliente en casi todos los campos en los que los de Mountain View han participado. Una vez más, han contribuido a aportar un servicio gratuito que se puede emplear sin ningún sobrecoste, pero con algunas limitaciones. Google Colab se trata de un servicio en línea basado en Jupyter Notebooks que nos permite emplear aceleración en GPU. Esta diseñado para diseñar aplicaciones de aprendizaje profundo, de ahí que haya utilizado hasta el momento unicamente Python 2.7 y 3.6. Adicionalmente se pueden instalar y emplear frameworks (2.3) como Pytorch, Keras, Tensorflow u OpenCV.

En mi caso, la utilidad anteriormente mencionada ha sido el motivo principal por el cual he empleado Google Colab. Esta herramienta se encuentra automáticamente ligada al Google Drive de la cuenta con la que accedes al servicio. Por tanto, es posible almacenar una gran cantidad de imágenes en la nube, para posteriormente usarlas desde Colab para entrenar modelos de aprendizaje profundo.

El punto positivo del desarrollo en Google Colab, se trata de la posibilidad de emplear una GPU dedicada y de gran potencia. Estas son las especificaciones más detalladas de la gráfica que emplea Google en este servicio:

**NVIDIA Tesla K80**

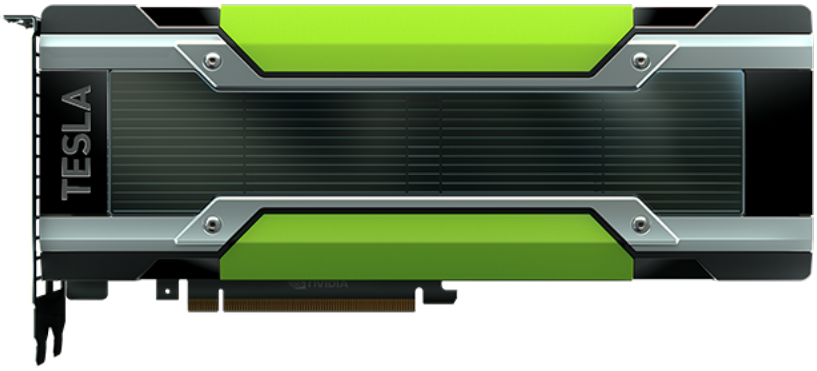



**Figura 2.7:** Tarjeta NVIDIA Tesla K80

- Arquitectura: Kepler
- Memoria: 24 GB GDDR5
- Núcleos: 4992 CUDA Cores
- Interfaz de memoria: 2 x 384-bits
- Ancho de banda de la memoria: 480 GB/s

Para obtener más información a cerca de esta tecnología, es interesante consultar la siguiente referencia [19].

## 2.3. Frameworks

Un framework, se trata de un conjunto de conceptos, prácticas y criterios necesarios para enfocar un tipo de problema particular que sirve para enfrentar y resolver nuevos problemas de índole similar. Es decir, estos frameworks recogen conjuntos de métodos que se han desarrollados anteriormente, testeados para garantizar su uso, y disponibles para realizar tareas similares a las establecidas por los desarrolladores, son como pequeñas bibliotecas de código y funcionalidades.

En el ámbito de la inteligencia artificial, disponemos de una gran cantidad de frameworks capaces de aportar funcionalidades vitales hoy día para el desarrollo de modelos de aprendizaje. Entre los más importantes, puedo destacar tres, los dos primeros se podría decir que son los referentes en este sector, y el tercero ha ido cogiendo peso en los últimos años, por este motivo ha sido el empleado para las tareas de entrenamiento automático.

### 2.3.1. TensorFlow

TensorFlow es una biblioteca de código abierto para aprendizaje automático desarrollada por Google con el objetivo de satisfacer sus propias necesidades. Estas se que concuerde el tiempo con la frase anterior según escojas de sistemas capaces de construir y entrenar redes neuronales para detectar y descifrar patrones y correlaciones, análogos al aprendizaje y razonamiento usados por los humanos. A pesar de ser una tarea bastante definida, el uso de esta librería se puede extrapolar a multitud de desarrollos diferentes. Un análisis del framework puede determinar de si se trata de una herramienta imprescindible en este tipo de prácticas.


- Pros del uso de TensorFlow
  - Gráficas dinámicas para el muestreo de datos.
  - Gestión de bibliotecas y gran cantidad de actualizaciones.
  - Capacidad de depuración de partes aisladas del código.
  - Gran escalabilidad para el uso en diferentes dispositivos hardware.
  - Altamente paralelo y diseñado para emplear multiples Unidad de Procesamiento Gráfico (GPU).
- Contras del uso de TensorFlow
  - Carencia de bucles simbólicos.
  - No dispone de soporte para Windows.
  - No dispone de soporte para tarjetas gráficas diferentes a NVIDIA.

Para obtener más información a cerca de esta tecnología, es interesante consultar la siguiente referencia [20].

### 2.3.2. Keras

Keras es una biblioteca de código abierto escrita en Python que es capaz de ejecutarse sobre TensorFlow, Microsoft Cognitive Toolkit o Theano. Fue diseñado para proporcionar una API de nivel superior a TensorFlow con el fin de facilitar y acelerar las experimentaciones, mientras se mantiene completamente transparente y compatible con él. Por tanto, podemos entender que Keras es un framework que emplea a su vez otros framewors. Se trata de una tecnología modular y extensible, con un aspecto amigable que permite desarrollar soluciones en poco tiempo.

Su propósito, su modularidad y extensibilidad y su capacidad de empleo más o menos sencilla lo convierten en una herramienta a tener en cuenta en el marco del aprendizaje automático. Pero también es muy importante conocer cuales son sus puntos negativos.

Estamos hablando de un framework que pretende abstraer al desarrollador de la complejidad que presenta TensorFlow (2.3.1). Por ende, cuando surge un problema originado a bajo nivel, el uso de Keras para encontrar el error se vuelve un auténtico caos. El otro factor negativo se trata de una falta de personalización a nivel de superficie. Es decir, a pesar de incluir las funcionalidades suficientes, en ciertas ocasiones sera necesario crear por nuestra cuenta capas simples personalizadas o funciones de perdida más sofisticadas o específicas.

Para obtener más información a cerca de esta tecnología, es interesante consultar la siguiente referencia [21].

### 2.3.3. PyTorch

PyTorch es una biblioteca de aprendizaje automático de código abierto para Python, basada en Torch. Su principal funcionalidad son aplicaciones como el procesamiento de lenguaje natural. Fue desarrollado principalmente por el grupo de investigación de inteligencia artificial de Facebook, y el software de lenguaje de programación Pyro.

Como características principales de PyTorch se pueden destacar las siguientes:

- Cálculo de tensores, similares a NumPy, con fuerte aceleración de GPU.
- Redes neuronales profundas construidas en un sistema autodiff basado en cinta.
- Facilidad de utilización.
- Mayor flexibilidad de desarrollo respecto a TensorFlow, lo que indica que en ciertas ocasiones puede resultar más potente.
- Gran facilidad para depurar el código.
- Gran capacidad de personalización, esta permite desarrollar tareas de investigación mucho más sencillas que con el resto de herramientas.

Para obtener más información a cerca de esta tecnología, es interesante consultar la siguiente referencia [22].

## 2.4. Resumen

Es muy importante partir de una base solida sobre la que construir nuestro conjunto de datos o dataset. Por ello una exhausta investigación y análisis sobre las acciones a desarrollar sobre un entorno familiar y cotidiano nos va a permitir mejorar la actuación de nuestro robot en este tipo de entornos. De modo que, a modo de repaso voy a enumerar los puntos más importantes a tener en cuenta si de unos cimientos sólidos se desea partir:

- Analizar las acciones más relevantes puede suponer un cambio sustancial en el desarrollo de un robot con conocimientos suficientes para desarrollar las tareas deseadas.
- Definir clases en función de estas acciones y desarrollarlas para poder grabarlas con mayor eficacia.
- Emplear equipo con la calidad suficiente como para capturar los movimientos establecidos en el análisis.
- Generar un conjunto de ficheros exportados y ordenados para facilitar el trabajo de etiquetado. Estas exportaciones deben estar configuradas correctamente, por ende es importante ajustar las rotaciones o frames por segundo de estas respecto al motor gráfico deseado.
- El etiquetado puede resultar la tarea más tediosa, pero su importancia es crucial para obtener datos verídicos. Emplear tiempo en resolver este apartado correctamente puede suponer un ahorro de tiempo con posibles errores en el entrenamiento de nuestro robot.

# Capítulo 3

# Analizando y generando acciones

*Como se ha comentado en apartados anteriores, el objetivo principal del proyecto consta de la construcción de un dataset sintético. Esta construcción viene dividida en diferentes puntos, que han permitido desarrollar una solución acorde a las necesidades iniciales planteadas. Durante el transcurso de este proceso, se podrán ver tanto el análisis del trabajo realizado como pruebas y ejemplos llevados a cabo.*

## 3.1. Introducción

Este primer apartado corresponde a la mayor parte del proyecto, en la cual se ha tenido especial atención al análisis de las acciones. Es importante comprender qué tipos de acciones se van a desarrollar y cómo se pueden segmentar. Garantizar una correcta calidad del dato permite evitar posibles errores o ambigüedades eludiendo su aparición en el proceso de entrenamiento.

Sin embargo, el resto del trabajo que ha implicado proyecto también ha sido muy importante, en concreto aquel que ha involucrado a Unreal Engine 4. La flexibilidad que este motor nos proporciona para trabajar con animaciones ha permitido el desarrollo de una solución propia que satisface los requerimientos del proyecto. Una vez etiquetadas estas animaciones, es posible emplear este software para desarrollar por mi mismo una solución acorde al proyecto. Sin salir de Unreal Engine 4, y añadiendo UnrealROX a la fórmula, fue posible grabar el resultado de unir la animación almacenada en formato BVH y el etiquetado de acciones, y generar posteriormente una gran volumetría de datos.

## 3.2. Análisis de acciones

Un buen edificio se empieza por los cimientos, y para que el edificio sea sobresaliente, estos cimientos deben ser sólidos. De esta misma forma me he tomado este apartado, el punto inicial es crítico para garantizar que los datos generados son, como mínimo, fiables.

Es necesario establecer una serie de acciones deseadas, las cuales se analizarán y grabaran mediante las herramientas nombradas en puntos siguientes. Para mi caso, las acciones seleccionadas han sido las siguientes: levantarse, caminar hacia delante, abrir armario/frigorífico, coger botella, abrir botella, beber agua, cerrar botella, dejar botella, cerrar armario/frigorífico, caminar hacia atrás y sentarse. Como se puede apreciar, estas acciones conforman una secuencia, llevándose a cabo una detrás de otra. La razón por la cuál se han seleccionado estas acciones es bien simple, buscaba comenzar con

acciones sencillas, sin demasiada complejidad, pero dotando de cierta interacción con el entorno. Lo que he tratado de evitar es buscar demasiada dificultad desde un inicio, una vez finalizado el proyecto, siempre es posible analizar y grabar nuevas acciones.

Estas clases son diferentes unas de otras, a pesar de compartir movimientos o comportamientos comunes unas con otras. Siendo más específicos, las clases de caminar hacia delante o hacia atrás, son iguales, a excepción de la dirección de movimiento. Este factor es de gran importancia para poder dotar al robot de diferentes métodos para realizar una misma acción, caminar en este caso. Cada acción, a su vez, se divide en subacciones, por tanto tiene sentido poder diferenciar estas dos acciones mediante la subdivisión de ellas mismas.

Las 11 clases establecidas, se diferencian todas unas de otras, pero es importante analizarlas por separado, para estar seguros de que constituyen unos cimientos sólidos para nuestro edificio.

- **Levantarse**

  Esta acción tiene su inicio en una silla sentados, y durante unos 100 frames comienza a levantarse del asiento para acomodarse en una postura estática. Los huesos implicados en el movimiento son sobre todo piernas, en concreto rodillas, y la zona de la cadera, que permite el movimiento hacia arriba de la espalda. Este movimiento se puede apreciar en la Figura 3.1.

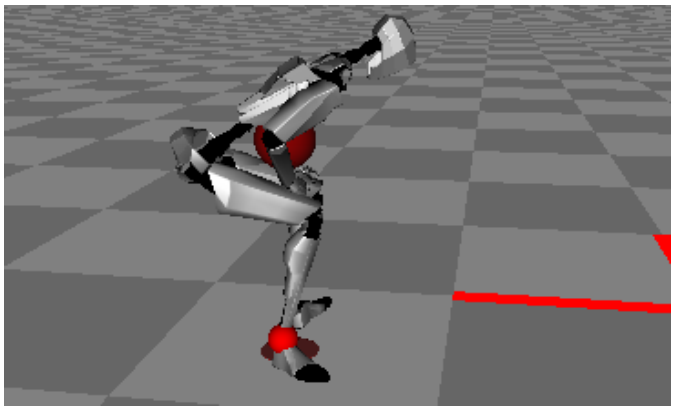


**Figura 3.1:** Movimiento de levantarse de una silla.

- **Caminar hacia delante**

  Esta acción tiene su inicio en una posición estática, y durante unos 100 frames comienza a caminar hacia delante para finalizar su movimiento al mover dos veces las piernas, una vez la derecha y otra vez la izquierda. Los huesos implicados en el movimiento son sobre todo las piernas y los brazos, los cuales van a tener un movimiento inverso en todo momento. Esto quiere decir que cuando se mueva la pierna derecha hacia delante, el brazo izquierdo será el encargado de realizar el movimiento hacia delante y viceversa. En todo momento es importante mantener una postura erguida, garantizando un correcto movimiento. Este movimiento se puede apreciar en la Figura 3.2.

- **Abrir armario/frigorífico**

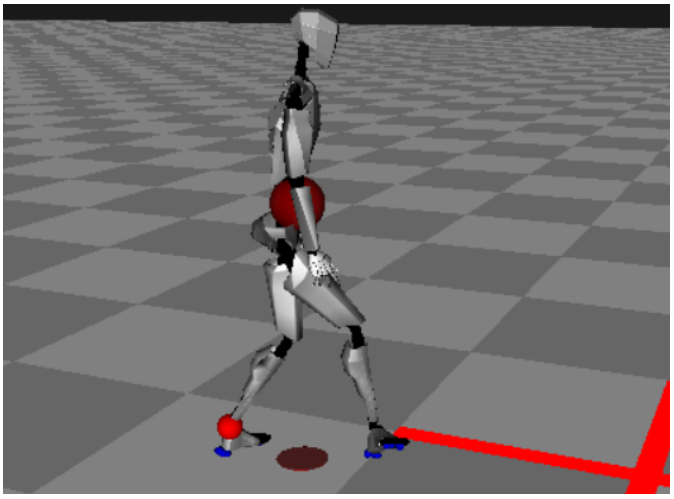


**Figura 3.2:** Movimiento de caminar hacia delante.

Esta acción tiene su inicio en una posición estática en frente de un armario o frigorífico, y durante un poco mas de 100 frames trata de abrir la puerta del mueble sobre el que se encuentra. Los huesos implicados en el movimiento son los correspondientes al brazo derecho o izquierdo, en función de la mano que se use. El movimiento se caracteriza por un desplazamiento circular de fuera hacia dentro, diferente de un movimiento recto como el de caminar. Esto va a permitir dotar al robot de diferentes tipos de movimientos en un mismo hueso, lo que a su vez permite obtener un dataset mucho más rico en acciones. Este movimiento se puede apreciar en la Figura 3.3.

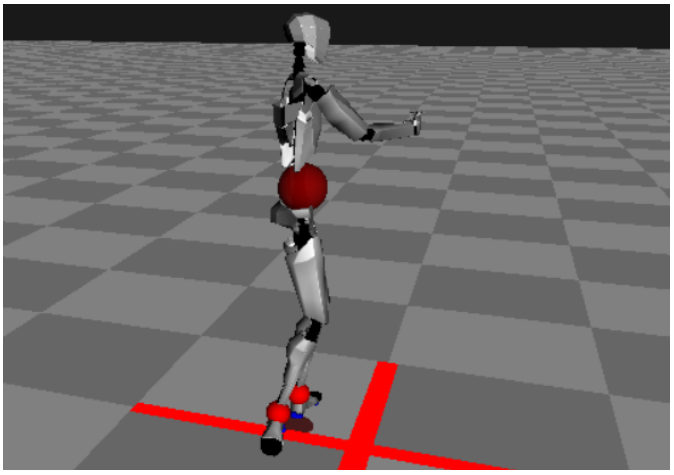


**Figura 3.3:** Movimiento de abrir un armario o frigorífico.

- **Coger botella**

  Esta acción tiene su inicio en una posición estática en frente de un armario o frigorífico, mientras se procede a coger con una de las manos un objeto del armario o frigorífico previamente abierto. La acción transcurre durante unos 150 frames en la que se extiende el brazo para coger con la mano el objeto. Los huesos implicados en el movimiento son los correspondientes al brazo derecho o izquierdo, en función de la mano que se use. En este caso, el movimiento se caracteriza por

un desplazamiento rectilíneo con una apertura del brazo hacia fuera. Esto va a permitir dotar al robot de la capacidad de poder extender el brazo para coger objetos más alejados. Este movimiento se puede apreciar en la Figura 3.4.

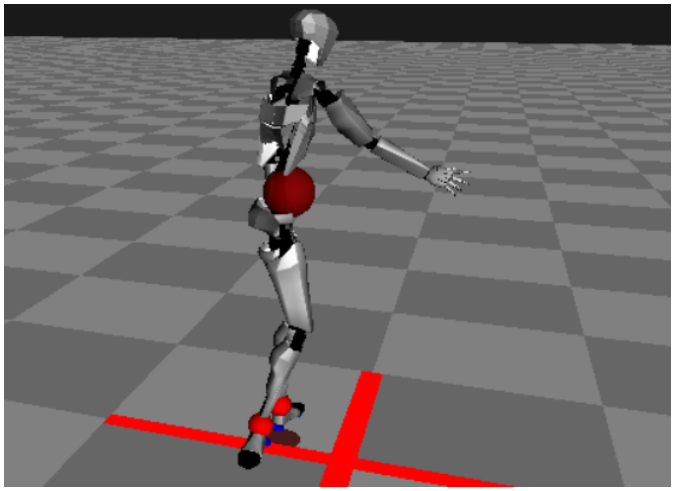


**Figura 3.4:** Movimiento de coger una botella u otro tipo de objeto.

- **Abrir botella**

  Esta acción tiene su inicio en una posición estática en frente de un armario o frigorífico, mientras se esta sujetando una botella con una de las manos, y con la otra se procede a retirar el tapón de la parte superior de la misma. La acción transcurre durante unos 150 frames en la cual el actor retira el tapón. Los huesos implicados en el movimiento son los correspondientes al brazo derecho o izquierdo, en función de la mano que se use. El movimiento se caracteriza por un desplazamiento circular de muñeca, la cual rota sobre su eje para destapar la botella. Esto va a permitir dotar al robot de habilidades con un grano de precisión mucho más fino, capaces de rotar huesos sin mover el resto del cuerpo. Este movimiento se puede apreciar en la Figura 3.5.

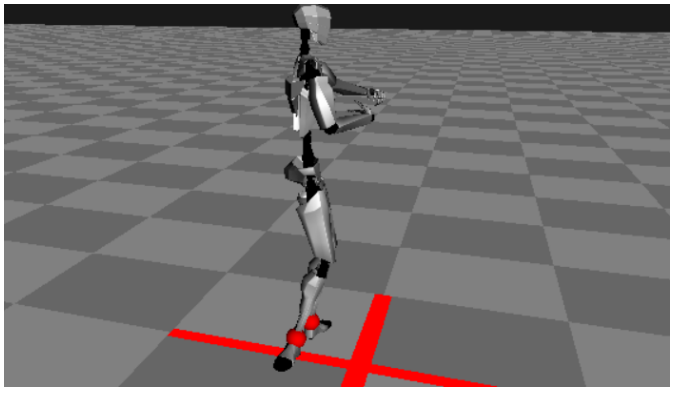


**Figura 3.5:** Movimiento de destapar el tapón de una botella.

- **Beber agua**

  Esta acción tiene su inicio en una posición estática, mientras se esta sujetando una botella con una de las manos, en la que dicha botella ya ha sido destapada. La acción transcurre durante unos 300 frames en la cuál el actor procede a beber

del contenido de la botella que sujeta. Los huesos implicados en el movimiento son los correspondientes al brazo derecho o izquierdo, en función de la mano que se use. El movimiento se caracteriza por un desplazamiento lineal hacia la cabeza, arqueando a su vez columna y cuello para favorecer a la acción. Este movimiento se puede apreciar en la Figura 3.6.

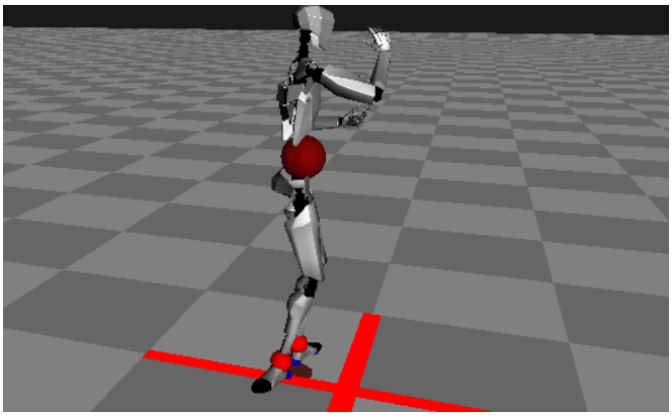


**Figura 3.6:** Movimiento de beber agua de una botella.

- **Cerrar botella, dejar botella, cerrar armario/frigorífico, caminar hacia atrás y sentarse**

  Para simplificar en estas acciones, las he agrupado en una única explicación. Esto es debido a que todas ellas son idénticas a las anteriores mostradas, pero con la particularidad del sentido de la acción. Todas estas tareas se desarrollan en orden inverso a las anteriores. Esto, llevado a la práctica, va a permitir a nuestro robot obtener la capacidad para hacer y deshacer acciones ya completadas, o simplemente aumentar la cantidad de movimientos disponibles en determinadas situaciones, disponiendo de un dataset más rico.

Estas clasificaciones han sido realizadas en función de un conjunto de tareas comunes en entornos familiares. La idea es obtener con pocas acciones, una gran cantidad de diferentes tipos de datos que dotará al robot de la capacidad suficiente para completar su tarea designada, ya sean por separado o enlazando diferentes clases unas con otras. Según lo descrito, podemos identificar diferentes tipos de movimientos principales y variados, entre los mas destacables están: movimiento coordinado de piernas y brazos, extensiones y contracciones de rodilla y codo, y rotación de brazo y muñeca.

## 3.3. Grabación de acciones

Teniendo en cuenta las clases definidas en la Sección 3.2, es necesario grabarlas por medio de herramientas destinadas a la captura del movimiento. Para ello, se ha empleado una tecnología llamada Perception Neuron (2.1.2), con una configuración capaz de obtener la posición y rotación de 59 huesos por medio de 32 neuronas. Cabe destacar que el esqueleto que incorpora Unreal Engine 4 en sus proyectos base consta de 59 huesos, los cuales concuerdan, aunque con nombre diferentes, con los 59 huesos exportados por medio de esta herramienta. Junto a esta configuración base del Perception Neuron, tenemos la correspondiente a la salida del software asociado al traje, el Axis Neuron (2.2). Esta configuración consta de una reproducción de 60 frames y una organización de las rotaciones por medio de la estructura YXZ en el eje de coordenadas

tridimensional.

Es posible realizar esta tarea por medio de únicamente 18 neuronas, aunque esto requiere una configuración adicional en Unreal Engine 4, ya que se generan errores al tratar de encontrar todas las neuronas. Esta diferencia de neuronas corresponde a las falanges entre los dedos de las manos, a cada una de estas se le incorpora una o dos neuronas. Este aumento en el número de neuronas empleado, permite obtener una mayor precisión en la actuación del sistema, contrarrestando el proceso con un aumento del tiempo necesario para generar este dataset sintético.

Teniendo en cuenta estas características, el objetivo fue grabar un conjunto de acciones correspondientes a las clases previamente definidas y exportarlas desde Axis Neuron en formato BVH con un orden YXZ para las rotaciones y un ratio de 60 Fotogramas por segundo (FPS). Para hacer esta tarea más sencilla, el proceso se centro en generar tres archivos de movimiento, en los cuales se incluía cada una de las clases mencionadas anteriormente, repetidas entre 5 y 10 veces cada una. ¿Por qué se han realizado las grabaciones repetidas veces? La respuesta es bien sencilla: cuando adquirimos productos artesanales de cualquier tipo, es improbable (por no decir imposible) encontrar dos iguales. Al igual que con los productos artesanales, podríamos llevar a cabo miles de grabaciones y que todas ellas difieran unas de otras. Esta acción dota al robot de la capacidad para comprender los pequeños cambios entre movimientos, como dice el refrán: todos los caminos llevan a Roma.

La precisión con la que se generan las grabaciones no es perfecta, como ocurre con casi todo en el medio físico. Este sistema cuenta con un pequeño margen de error, el cual no va a suponer un impedimento para desarrollar la solución final. La razón principal por la que este error puede ser compensado, es la posibilidad de cambiar las mallas del esqueleto en varias generaciones del dataset. Esto quiere decir que cambiando las mallas y rotando ciertos huesos se tratará de diversificar este dataset. Igual que anteriormente se hablaba de la diferencia de actuación entre diferentes ejecuciones de una misma acción, se puede aplicar esto mismo a la malla del esqueleto, no todas las personas somos físicamente iguales.

Teniendo claramente definidas las características de la grabación y del alcance que deseamos lograr, solamente resta conectar el traje del Perception Neuron con el ordenador y con el Axis Neuron por medio de USB. Estas grabaciones se van a almacenar en formato raw, es decir, almacena en el archivo los datos en crudo. Para poder trabajar con estos datos, es necesario convertirlos en formato BVH, esta tarea es rápidamente realizada en el propio Axis Neuron, el cual me permite extraer los frames deseados del fichero en lugar de todo el contenido. Esta es la forma que se ha empleado para separar las clases grabadas en cada uno de las grabaciones. Otra opción contemplada, fue exportar las acciones en formato FBX, para importarlo por medio de blender (2.2.3) y exportarlo a BVH. Esto último podría parecer que no tiene sentido, si no fuera por un detalle: en ocasiones y con ciertas mallas, los nombres del hueso padre del esqueleto empleado y de la animación, al ser diferentes, pueden provocar errores. Esto se solventa empleando blender para cambiar el nombre de este hueso.

En esta representación, mostrada en la Figura 3.7, se puede apreciar cuál es la jerarquía que ha empleado Axis Neuron para componer el esqueleto en la animación. El

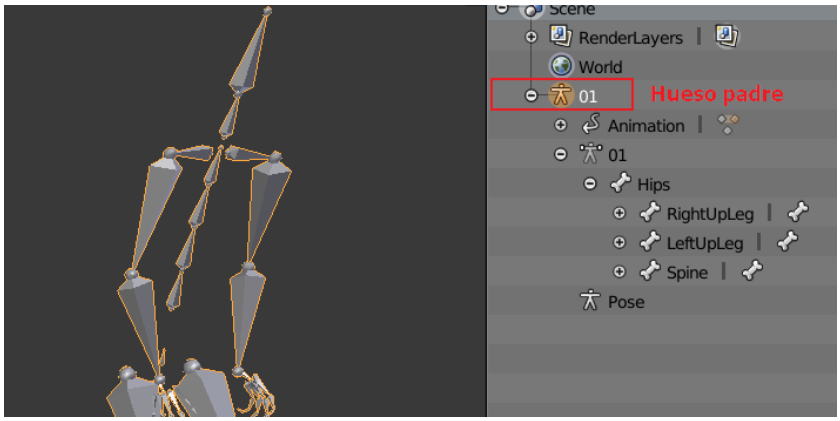



**Figura 3.7:** Distribución de huesos de la animación de levantarse.

primer componente se trata de un objeto de tipo Armature, que es usado por el software para almacenar todos los huesos del esqueleto. El archivo de salida tiene el nombre de 01.bvh, y el componente Armature adquiere el nombre de 01, es decir, se corresponde al nombre del fichero. Cuando se importa esta animación se exporta a Unreal Engine 4, se le asigna a este componente la posición correspondiente al hueso root.

Esto en algunas ocasiones y para ciertas tareas, puede generar problemas. Esta situación se da sobre todo si se emplea el plugin oficial proporcionado por los propios desarrolladores de Axis Neuron [17]. Este plugin es muy útil si lo que deseamos es reproducir la animación que hemos grabado, sobre el propio esqueleto del Axis (lo podemos encontrar en un proyecto demo junto con el plugin) o sobre el esqueleto del proyecto base de Unreal Engine 4. En el segundo caso, la tarea es algo más tediosa, ya que se debe realizar una tarea denominada como *retargeting* [23], [24], en el Capítulo 4 se profundizará más en esta tarea.

## 3.4. Etiquetado

Estas animaciones generadas con el Perception Neuron, exportadas en el formato adecuado e importadas en Unreal Engine 4 deben ser etiquetadas para determinar qué acción se está generando en cada uno de los frames. Para enterder mejor esta idea voy a simplificar el estado actual, es decir, vamos a pensar que solamente disponemos de dos clases. Las clases que vamos a emplear son: levantarse y caminar hacia delante, las cuales se tratan de acciones padres con subacciones asociadas. El concepto se visualiza en la Figura 3.8.

En el diagrama se aprecian dos niveles de abstracción. El primer nivel corresponde a las clases comentadas en el apartado 3.2, y por debajo de estas un conjunto de subacciones, las cuales pueden aparecer en más de una clase padre. Además de estos dos niveles de abstracción, aún se podría añadir un tercero: el nombre de la actividad, actuando como padre de las acciones. Si por ejemplo hiciera referencia a la actividad de beber agua, esta consistiría en múltiples clases: caminar, abrir el frigorífico, coger la botella, beber... Y cada una de ellas se compondría de una serie de movimientos concretos más simples como pueden ser mover pierna derecha/izquierda, levantar el brazo derecho/izquierdo, girar la muñeca... Sin embargo, a pesar de haber descrito este tercer nivel de clasificación, no lo vamos a tratar en este documento, ya que el ámbito tratado

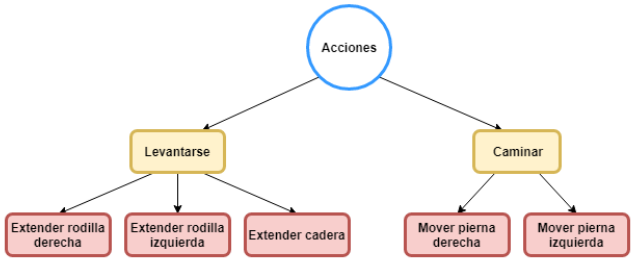



**Figura 3.8:** Distribución de acciones y subacciones de clases.

se escapa del alcance del proyecto.

Volviendo al diagrama mostrado, disponemos de nuestras dos clases: levantarse y caminar, con sus subacciones correspondientes. El objetivo es realizar un etiquetado de clases y subacciones por frames, de tal forma que para cada uno de los frames de la animación, podamos saber cuál es la acción y, dentro de ella, la subacción, que se están llevando a cabo. Si analizamos la acción como una secuencia de estas subacciones de cada clase, podríamos comprender, y mas importante aún, hacer comprender a nuestro robot qué ocurre en esta sucesión de acciones. Al dotar de este conocimiento al robot, puede tomar cierta percepción de lo ocurrido y tomar decisiones más ajustadas a la realidad (1.3.2).

La creación de un script capaz de generar un fichero Notación de objeto JavaScript (JSON) con el etiquetado de cada una de las animaciones ha sido crucial para ahorrar tiempo en este punto. Para este desarrollo se ha utilizado el lenguaje de programación Python (2.2.1) debido a su versatilidad y compatibilidad con diferentes sistemas operativos, además de su fácil manejo para este tipo de tareas. El script es semiautomático, ya que es preciso seleccionar las acciones y subacciones frame a frame. Las funcionalidades dotadas van desde seleccionar la ruta y fichero deseado, hasta etiquetar el fichero y visualizar el contenido del archivo JSON generado. El uso de este script debe complementarse con otro software capaz de ejecutar frame a frame las animaciones, ya que no dispone de ningún tipo de reproductor. En mi caso, la opción más factible ha sido emplear Blender (2.2.3), ya que permite un control total sobre la secuencia de frames correspondientes a la animación. Al finalizar la tarea, vamos a obtener un fichero con una estructura muy similar al código (3.1)

La distribución indicada se puede ver representada en tres niveles diferentes, en el cual dicho nivel principal corresponde a un tipo genérico, ya que en este proyecto solo vamos a trabajar con dos niveles de abstracción. En los otros dos niveles se distribuyen las clases y subacciones correspondientes y dentro de cada una de estas, los frames. Posteriormente estos datos van a servir para alimentar el sistema de generación de datos del que dispone UnrealROX para procesar y generar un dataset completo, teniendo como salidas las imágenes obtenidas por las diferentes cámaras y los movimientos concretos de cada frame. Para obtener más información acerca del código empleado en el script es preciso consultar el apéndice A.

```
{
  "Generico": {
    "Levantarse": {
      "Estatico": [
        1,2,3,4,5,6,7,8,9,10,11,12,13,14,15...
        ],
      "Inclinarse_hacia_delante": [
        26,27,28,29,30,31,32,33,34,35,36,37...
      ],
      "Inclinarse_hacia_atras": [
        79,80,81,82,83,84,85,86,87,88,89,90...
      ],
      ...
    }
  }
}
```

**Listing 3.1:** Ejemplo de fichero BVH

## 3.5. Resumen

En este capítulo se pueden recoger diferentes aspectos relevantes a la hora de desarrollar una tarea desde un aspecto analítico. Es muy importante partir de una base solida sobre la que construir nuestro conjunto de datos o dataset. Por ello una exhausta investigación y análisis sobre las acciones a desarrollar sobre un entorno familiar y cotidiano nos va a permitir mejorar la actuación de nuestro robot en este tipo de entornos. Esto, unido a un hardware de gran precisión y a una actuación correcta de las animaciones por parte del usuario, va a permitir obtener los datos del escenario de forma fiable. Posteriormente se generarán ficheros con el etiquetado que nos permitirá a nosotros y al robot saber a que movimiento corresponde cada uno de los frames.

# Capítulo 4

# Integración con Unreal Engine

*Partiendo del paso anterior, debemos tener nuestras clases grabadas y disponibles, junto con un fichero por clase que contiene el etiquetado de estas acciones. Esto es la entrada que le proporcionaremos al motor gráfico para realizar los procesos comentados a continuación. Este apartado trata más a fondo temas relacionados con Unreal Engine 4, como pueden ser desarrollos en C++, blueprints [1], mallas y esqueletos o rotaciones y traslaciones de huesos.*

## 4.1. Introducción

Generar el dataset es el objetivo principal a abordar con este proyecto, pero para ello debemos analizar antes cada uno de los pasos previos. Antes de poder generar datos, es preciso obtener por medio de un plugin la capacidad para leer, tratar, transformar y procesar los datos almacenados en los ficheros BVH. En primera instancia, me apoyé en plugins ya desarrollados, los cuales hago referencia en el Capítulo 5. Tras realizar los experimentos que en dicha sección se detallan con los plugins mencionados, se llegó a la conclusión final de desarrollar un plugin propio, ya que los primeros no cumplían con las necesidades básicas en cuanto a funcionalidades.

Además entraremos en temas más concretos de Unreal Engine como son el control de animaciones y los objetos que maneja este motor. Este es un apartado clave si se precisa comprender en profundidad el funcionamiento del motor y como controlar nuestros propios esqueletos por medio de las técnicas comentadas hasta el momento y las que le suceden.

## 4.2. UnrealROX

Como punto de partida nos encontramos con un proyecto realizado por la Universidad de Alicante [14] que nos permite generar el *ground truth* [2] de las grabaciones tomadas desde Unreal. Disponemos de un modelo que representa un robot humanoide para interaccionar con el entorno, y de diversas cámaras colocadas a nuestro gusto a lo largo de la escena. Estas cámaras son fácilmente colocadas por medio del editor de escena que proporciona Unreal Engine 4. Cabe destacar que este proyecto es open-source, por tanto es posible ajustar las características de este a las necesidades personales de cada usuario.

[1] Blueprints son el sistema de scripting visual dentro de Unreal Engine 4, que permite empezar a crear prototipos de juegos de forma rápida.

[2] Conjunto de segmentación de instancias, mapas de normales, profundidad y datos RGB


**Figura 4.1:** Empleando el grasping se dota al robot humanoide de la capacidad para agarrar de forma real objetos del entorno.

El propósito principal de UnrealROX es la generación del ya mencionado ground truth, que lo posiciona como una herramienta muy interesante a la hora de generar dataset basados en realidad virtual. Además podemos encontrar otras muchas caracterísiticas, las cuales han resultado ser un factor determinante para emplearlo como punto de inicio sobre el que desarrollar mi dataset. Entre las características más importantes hayamos un sistema de grasping [3] que permite agarrar objetos de la forma más real posible (4.1), aporta la posibilidad de incluir tantas cámaras como se desee ya sean estáticas o dinámicas, nos permite emplear las grabaciones para proyectos en primera o tercera persona. Además, al tratarse de código abierto permite una gran personalización, haciendo especial mención en a la posibilidad de elegir entre foto-realismo o aletorización de dominios como técnicas de realidad virtual para la captura de imágenes.

## 4.3. Plugins de control de animaciones

Podemos encontrar y acceder a múltiples plugins que nos permiten controlar los archivos BVH desde el motor gráfico seleccionado para este desarrollo, aunque las limitaciones de cada uno de estos son diferentes. Dependiendo del propósito que deseemos, vamos a necesitar emplear un plugin u otro. El camino recorrido para encontrar el plugin ideal pasa la integración del uso de estos ficheros con UnrealROX (4.2) fue necesario experimentar con diversos métodos, a continuación se van a explicar los más relevantes. Todos los experimentos llevados a cabo con cada uno de ellos se pueden consultar en el Capítulo 5.

- **Perception Neuron Unreal Plugin**. Desarrollado por Perception Neuron [4], nos da la posibilidad de emplear archivos BVH para animar nuestros modelos previamente grabados con el hardware previamente empleado (3.3), desde la versión 4.14.1 hasta la 4.21.1 de Unreal Engine 4 (en el momento de la redacción, la versión 4.21.1 era la última, lo cual indica que los autores del plugin lo van actualizando según se publican nuevas versiones del motor). Debido a su relación directa con

[3] https://github.com/3dperceptionlab/unrealgrasp
[4] https://neuronmocap.com/content/unreal-plugin

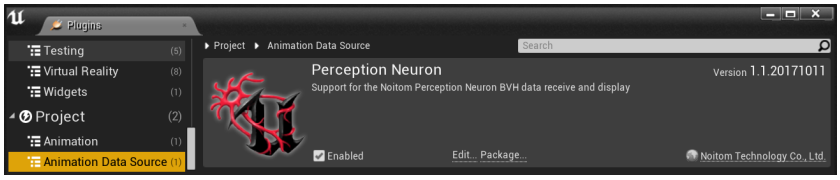



**Figura 4.2:** Plugin capaz de emplear ficheros bvh para reproducir animaciones.

el traje, fue la primera opción a tener en cuenta para trabajar con este tipo de ficheros. La integración con Unreal Engine 4 se basa en la importación del plugin al directorio correspondiente en el proyecto deseado. Una vez importado en nuestro proyecto podremos comprobarlo accediendo a través de **Edit->Plugins**(4.2), las características de este son:

- Permite importar un archivo BVH, que empleará de forma automática un *uasset* [5] para la reproducción del fichero sobre la malla deseada. Se puede controlar la velocidad de la animación empleando el propio editor de Unreal.
- Permite emplear el Protocolo de datagramas de usuario (UDP) para recibir una animación que está siendo reproducida en el software de captura, y aplicarla en tiempo real a un malla con esqueleto en la escena de Unreal.
- Permite reproducir la animación sobre diferentes esqueletos, siempre que dispongan de una pose inicial en forma de T. Para poder emplear esta técnica es necesario realizar *retargeting* a los huesos del esqueleto a emplear.
- Permite recibir por UDP los datos procedentes de varios servidores al mismo tiempo.

Empleando este plugin se logró reproducir cualquier animación en cualquier esqueleto deseado, unicamente incluyendo el plugin en el proyecto e importando el fichero deseado. A pesar de disponer de gran cantidad de posibilidades, únicamente podemos reproducir la animación variando el fichero incluido y el esqueleto empleado. La generación de un *uasset* a partir de un BVH es un factor que facilita en gran medida trabajar con este tipo de ficheros, los cuales no son reconocidos por Unreal. Cabe destacar que esto es algo que no se tiene en cuenta en ningún otro plugin, aunque como se verá más adelante no será necesario en el desarrollo final de la solución (4.3).

El principal problema que presenta es la carencia de la posibilidad de poder ejecutar frame a frame nuestra animación. Si no hay forma de realizar esta tarea, no es posible emplear el proceso de generación de ground truth característico de UnrealROX. La falta de esta funcionalidad propició el descarte de este plugin para leer los ficheros de grabación.

- **Perception Neuron Template Plugin**. Se trata de una plantilla y plugin no oficial desarrollado bajo la licencia del Instituto de Tecnología de Massachusetts (MIT)

[5] Es un componente de Unreal que contiene un asset, como pueden ser niveles, materiales, animaciones...

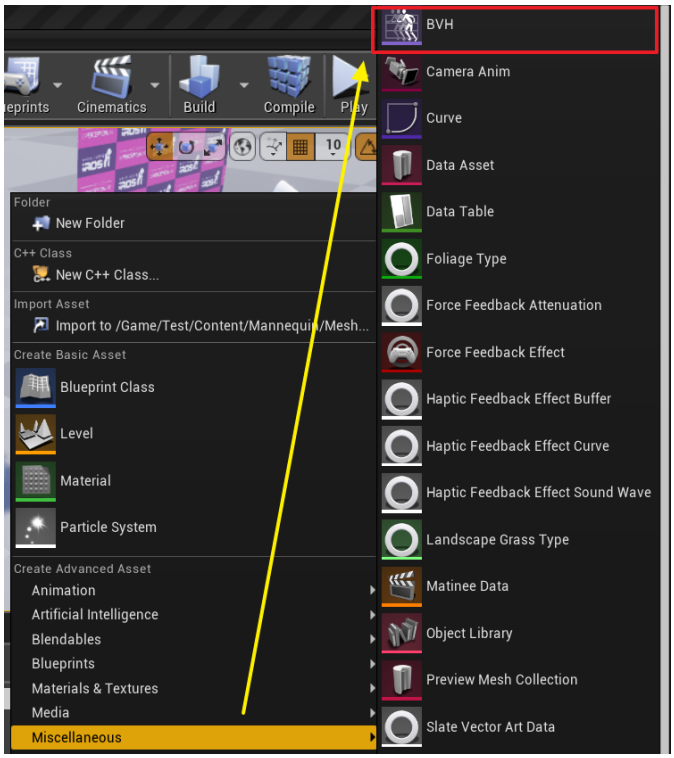



**Figura 4.3:** Selección del componente capaz de trabajar con los ficheros BVH.

para proyectos en tercera persona [6]. Las versiones sobre las que se puede emplear son algo más reducidas que en el plugin anterior, estando estas comprendidas entre las 4.10 y 4.18, siendo esta última la correspondiente a la empleada en este proyecto. Las posibilidades empleando este template son mayores que empleando el plugin oficial distribuido por Perception Neuron, entre las características adicionales más importantes podemos encontrar las siguientes:

- Permite la conexión directa al software Axis Neuron (2.2) por medio del protocolo Protocolo de control de transmisión (TCP).
- Lectura y almacenamiento de los datos del fichero, permitiendo su reproducción, pausa y rebobinado.
- Aporta el soporte necesario para trabajar con esqueletos que tengan poses en forma de T, A o con eje Y negativo.
- Grabación de la animación en el juego o en el editor (esta parte no pertenece a la versión gratuita).

Realmente se puede tratar como una extensión del plugin oficial, ya que realiza las mismas tareas y de forma similar, pero con estas características adicionales. La forma de tratar los ficheros BVH también es diferente, ya que no emplea *uassets* para trabajar con ellos. La lógica principal en la que se realiza la lectura del

[6] https://forums.unrealengine.com/development-discussion/animation/56791-perception-neuron-template

```
1  void APerceptionNeuronController::Tick(float DeltaTime)
2  {
3    Super::Tick(DeltaTime);
4
5    // Read each tick a new motionline. Additionally motionlines are
         just discarded.
6    if ((bConnected == true) || (bPlay == true))
7    {
8    ...
9    }
10 ...
11 }
```

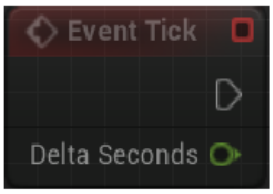



**Figura 4.4:** Función *Tick* de Unreal Engine 4 junto con el código base.

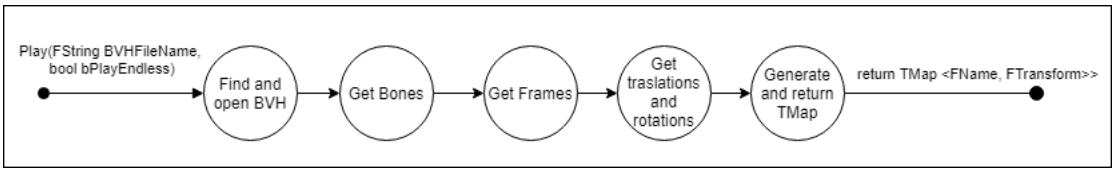



**Figura 4.5:** Diagrama de flujo que simplifica la actuación del método *Play*.

fichero y la reproducción de la animación empleando la función *Tick*[7] (4.4).

Para poder almacenar los datos guardados en el fichero de animación se emplea una función que lo recorre (4.5), este método se encarga de obtener el árbol en el cuál se estructuran los huesos, de igual forma se vió en la Sección 3.3. Recopila este esqueleto en una estructura de datos y también almacena los datos correspondientes al número de frames totales de la animación y de frames por segundo. Por último se procede a recorrer cada uno de los datos que contienen las rotaciones y traslaciones de cada uno de los huesos del esqueleto. En el fichero BVH se guarda una fila por frame, de cada fila cada seis valores corresponden a un hueso, de estos seis valores tres son de los vectores de traslación y tres de rotación, y cada uno de estos tríos se trata de los valores X, Y y Z de estos vectores.

De este modo, teniendo almacenados los datos necesarios, en cada ejecución del tick se realiza la aplicación de estas traslaciones y rotaciones al esqueleto seleccionado. El método tick corresponde a una función que se realiza en cada frame de juego, no de animación. En Unreal, el control de los frames de las animaciones estan separados del control de frames del juego, ya que estos últimos no son estables, dependen de las necesidades computacionales de cada instante y del hardware disponible para ese procesamiento.

Por tanto a pesar de incluir esta lógica para que se reproduzca en cada instante del juego, se incluye una técnica denominada como *Skip Frames* la cual permite ajustar la reproducción de la animación a la velocidad del juego en cada instante.

[7]Es una función que se puede sobrescribir al heredar de Actor, y que se ejecuta una vez por frame renderizado.

Esto se realiza por medio del valor DeltaTime, que contiene el tiempo que transcurre entre frames del juego expresado en segundos (4.1). Es decir, si tuviéramos una situación idónea en la que no se producen subidas ni bajadas de FPS (esto es una situación imposible en Unreal Engine 4 por el diseño de su motor, siempre hay mínimas variaciones casi imperceptibles, pero supongamos este escenario), y nuestra velocidad de reproducción es de 60 FPS, nuestro *DeltaTime* tendría un valor de 0,01667 segundos.

$$\begin{aligned} 1segundo/120frames &= 0,00833segundos \\ 1segundo/60frames &= \ 0,01667 \ segundos \\ 1segundo/30frames &= 0,03333segundos \end{aligned} \tag{4.1}$$

Empleando este *DeltaTime* y los frames por segundo de nuestra animación es posible calcular la cantidad de frames de animación que se deben reproducir respecto a los frames del juego. Supongamos de nuevo esta situación ideal de 60 FPS de juego estables, y grabamos una animación a 30 FPS. El resultado empleando esta técnica será la reproducción de 2 frames de animación por cada frame de juego. Esto es de gran utilidad para evitar los tirones que pueden aparecer como consecuencia de este desfase en los valores de frecuencia de ejecución de ambos flujos.

```
1  ...
2    DeltaTimeAdded -= FrameTime;
3    while (DeltaTimeAdded > FrameTime) // We are too slow => Skip frames
4    {
5      DeltaTimeAdded -= FrameTime;
6      MotionLinePointer++;
7      if ((MotionLinePointer >= (MotionLineOffset + Frames)) && (
           bEndless == true))
8        MotionLinePointer = MotionLineOffset;
9    }
10 ...
```

A pesar de incluir una gran cantidad de funcionalidades y características que se pueden emplear para la ejecución, pausa y rebobinado de estas animaciones, sigue siendo insuficiente para el propósito específico de este proyecto. De nuevo sigue sin poderse ejecutar de forma individual cada frame, o siendo más específico, ejecutar frame a frame las animaciones. La solución propuesta llegados a este instante fue modificar este template para tener las funcionalidades de reproducción ya existentes añadiendo unicamente la característica deseada.

- **Perception Neuron Template Plugin (Modificación Personal)**. La alternativa más sencilla resultó ser editar el template anterior para dotarlo de las capacidades deseadas, en concreto reproducir frame a frame la animación, incluso poder acceder al frame deseado en cada instante. Para comenzar con esta modificación, se creó una nueva clase en C++ llamada *NextFrame* y una clase en blueprint que llama al método en C++, de esta forma se puede desarrollar en C++ pero empleando estas clases desde el editor de Unreal Engine 4.

  En el segundo plugin, correspondiente al Perception Neuron Template proporcionado, podemos ver múltiples variables de control, entre todas ellas se localizó la encargada de controlar la reproducción de la animación, esta variable es: bool

bPlay. Cuando tiene el valor true se reproduce la animación, por el contrario cuando su valor es false se detiene. Por lo tanto, la planificación de esta modificación se centró en el control de esta variable por medio de pequeños cambios en el tick del juego.

El primer cambio fue eliminar el *Skip Frames* comentado anteriormente, ya que esta técnica es útil al reproducir de forma continua los frames de la animación, pero en este caso es innecesario ya que cada frame se va a reproducir cuando nosotros lo deseemos. Posteriormente se añadió un contador que permite controlar la cantidad de veces que el tick se ejecuta para poder controlar la variable de control y a su vez la animación. El planteamiento era acertado, pero el resultado no lo fue tanto. Cuando se intentaban controlar estas ejecuciones, en las primeras vueltas del tick la animación no se movía debido a operaciones de preparación que el *template* realizaba. Dichas operaciones no se podían suprimir por la estructura que se estableció, sería más sencillo realizar este proceso desde cero. Aunque esto resultaba un problema, aún se encontraba otro más preocupante: en diferentes ejecuciones del mismo algoritmo la cantidad de frames que se reproducían era diferente, bien por el desfase con los frames del juego o por posibles errores en el control del flujo. Intentando evitar perder demasiado tiempo con estos problemas se tomó la medida de desarrollar una solución propia capaz de realizar la tarea deseada.

- **BVH Player Unreal Plugin**. Este plugin o template consta de dos partes, la primera es una clase en C++ en la cuál se almacena todo el código encargado de tratar el fichero de animación. La segunda parte es la lógica desarrollada mediante blueprints para la lectura de estos datos.

  Para comenzar se debe entender el funcionamiento de este plugin. Su propósito es obtener los datos en crudo desde un fichero BVH y almacenar en diferentes tipos de estructuras de datos el conjunto de huesos del esqueleto (4.1), el recuento de frames de la animación y la velocidad de esta, y cada uno de los datos de traslación y rotación de los huesos (4.2). Como se puede apreciar la idea es exactamente igual que la proporcionada por los desarrolladores del Perception Neuron Template, con la peculiaridad de eliminar la lógica de la función tick de Unreal. Es decir, se dispone de un método encargado de leer el fichero y almacenar los datos.

  Es preciso tener en cuenta que cada hueso contiene un offset (4.1) que es necesario añadir a la traslación del hueso en el esqueleto, ya que sin estos valores no sería posible componer correctamente la malla. En cuanto a los valores de movimiento de la animación (4.2), estos corresponden con la X, Y y Z de traslación y el Pitch Roll y Yaw respectivamente. En el código solo se aprecian seis valores, pero hay seis por cada hueso que empleemos, y una línea por cada frame de animación. Con estos datos podemos construir nuestra secuencia de movimiento, cargarla en memoria y reproducir los datos frame a frame.

```
1  HIERARCHY
2  ROOT Hips
3  {
4    OFFSET 0.000 93.594 0.000
5    CHANNELS 6 Xposition Yposition Zposition Yrotation Xrotation
         Zrotation
6    JOINT RightUpLeg
7    {
8      ...
9    }
10   ...
11 }
12 ...
```

**Listing 4.1:** Estructura de huesos establecida por los ficheros BVH.

```
1  ...
2  Frames: 105
3  Frame Time: 0.017
4  -145.988510 93.153046 52.607071 94.739563 17.655418 5.136321
5  ...
```

**Listing 4.2:** Velocidad, cantidad de frames y porción correspondiente a la traslación y rotación.

Cuando se leen los datos en crudo, se formatean como FString [8] y es necesario convertirlos en los tipos de datos correctos y almacenarlos en las siguientes estructuras:

- Huesos: TArray<FName>
- Traslaciones: FVector
- Rotaciones: FRotator
- Datos de todos los frames: TArray<TMap<FName,FTransform>>

Establecer estas estructuras se debe a la necesidad de emplear los componentes de FVector [9] y FRotator [10] para actualizar los datos de animación para nuestro actor. Estas estructuras se van a emplear por medio de blueprints, pero aún falta comentar un método más desarrollado este se encarga de cargar el siguiente frame cuando nosotros lo deseemos, para ello se ha definido una tecla del teclado que controla esta ejecución, en nuestro caso la tecla K. El código desarrollado se incluye en el apéndice B.

A nuestra escena es necesario incorporar una clase que herede de character y nuestra clase BVH Player [11] que contiene el código que permite leer y reproducir la animación. De forma adicional también es recomendable incluir un player controler y un game mode base para obtener un mejor manejo del movimiento

[8] Se trata de un string capaz de ser dimensionado dinamicamente empleado por Unreal Engine 4.
[9] Struct de Unreal que representa la posición con tres *floats*: X, Y y Z.
[10] Struct de Unreal que representa la rotación con tres *floats*: Yaw, Pitch y Roll.
[11] Character se trata de una clase propia de Unreal Engine 4, BVH Player es la clase en C++ generada anteriormente, BVH Player BP es la clase generada en blueprints.

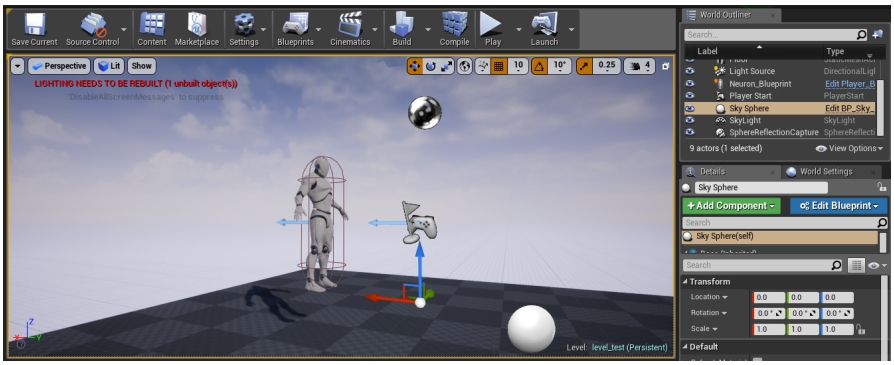



**Figura 4.6:** Vista de la escena básica para reproducir la animación.

en tiempo de ejecución y un control de la entrada por teclado lo más desacoplado posible respecto al nivel o escena. Una vez incluidos estos elementos en la escena, deberíamos tener una vista similar a la Figura 4.6. Una vez disponemos de esta representación, debemos ajustar la lógica a nivel de blueprints para ejecutar el código en C++.

- BVH Player BP: En primera instancia es necesario hacer una llamada al método encargado de recorrer el fichero de animación y almacenar los datos en las estructuras descritas con anterioridad, esta lógica se aprecia en la Figura 4.7. Para ello debemos obtener la instancia del character situado en la escena, al disponer solo de uno empleamos el método *Get All Actors Of Class* empleando como entrada la clase buscada. Posteriormente se hace la llamada al método *Play BVH File*, que recibe como entrada el nombre del fichero BVH, una referencia al objeto que hace la llamada y la instancia del character al que le aplicaremos las trasformaciones y rotaciones.

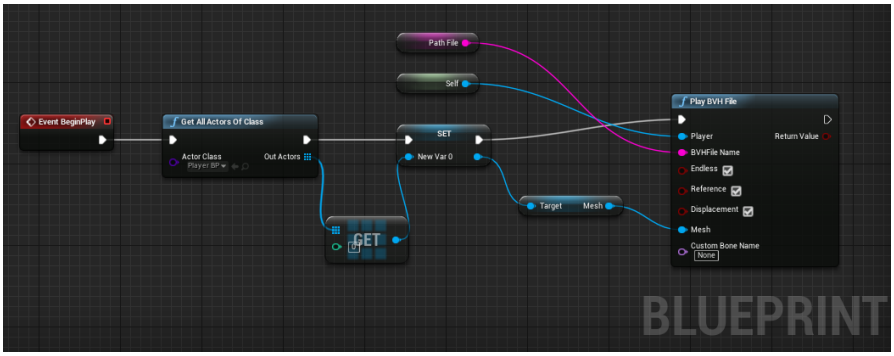



**Figura 4.7:** Lógica de blueprints necesaria para leer datos del bvh.

Cuando pulsamos la tecla K del teclado se activa el evento establecido como *InputAction NextFPS* situado en el Player Controller (siguiente apartado). Tras activarse este evento, a su vez llama al evento *Remote Key Player BVH* situado en el BVH Player, este evento se puede apreciar en la lógica de dicho objeto, mostrado en la Figura 4.8. Acto seguido se produce una llamada al método *Next Frame BVHFile* encargado de obtener el siguiente frame para la animación respecto al estado actual. Como salida devuelve el TMap con los

datos correspondientes a las traslaciones y rotaciones del frame encontrado. Es necesario enviar estos datos al objeto Player BP por lo que se realiza una búsqueda de este objeto en la escena y se actualiza el valor de la variable *Current Frame*.

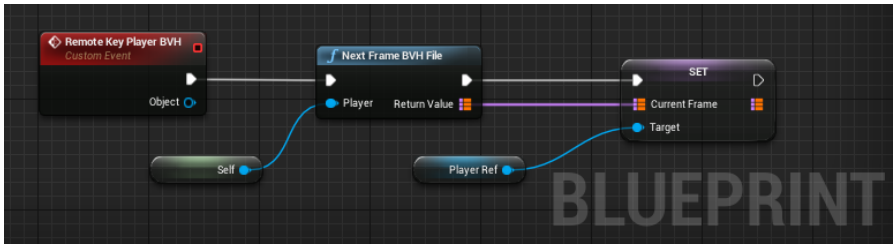



**Figura 4.8:** Lógica de blueprints necesaria para seleccionar el frame correcto para la animación.

- Player Controller: En mi caso este objeto contiene la lógica necesaria para desacoplar la pulsación de la tecla K respecto a la escena. Si se deseara prescindir de este objeto, es necesario incluir una lógica similar en el blueprint del nivel, lo que hace que dependa del nivel actual y no del objeto en cuestión. Al iniciar la ejecución este objeto obtiene la instancia del objeto BVH Player al que estamos observando para realizar la llamada al evento personalizado, Figura 4.9. Cuando se realiza la pulsación de la tecla seleccionada, se activa el evento *InputActiojn NextFPS* que realiza la llamada del evento alojado en el objeto BVH Player, Figura 4.10, activando el método para seleccionar el siguiente frame, visto en la Figura 4.8

- Player BP: Es el componente que contiene la malla del personaje y el esqueleto al que aplicar las transformaciones correspondientes. Esta generado en blueprints, hereda directamente de Character y no se aplica en su interior ninguna lógica. Unicamente se ha creado una variable visible de tipo TArray<TMap<FName,FTransform>> para almacenar los valores de traslación y rotación correspondientes al frame actual. Es necesario esta variable ya que el blueprint de animación obtiene los datos a partir del blueprint asociado a la animación, es decir, el Player BP en este caso.

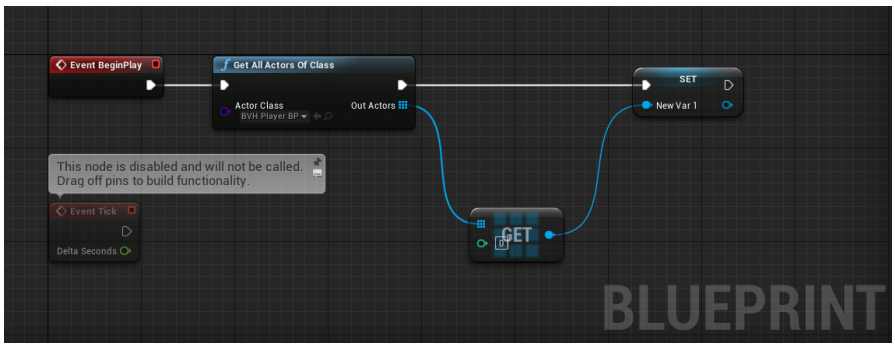



**Figura 4.9:** Lógica de blueprints necesaria para activar el evento personalizado al pulsar la tecla K.

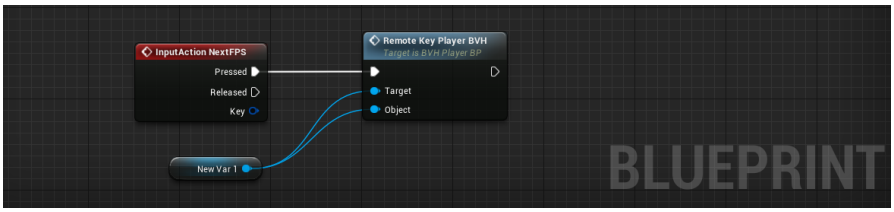



**Figura 4.10:** Llamada al evento personalizado *Remote Key Player BVH*.

- AnimNode: Se trata de un plugin desarrollado por la Universidad de Alicante, que ha sido empleado en el proyecto de UnrealROX [12]. La función que desempeña es la realización de las transformaciones correspondientes a los huesos del esqueleto asociado al blueprint de la animación. En la Figura 4.11 se puede apreciar el método de empleo de este componente, como entrada debemos pasar una transformación en la que se incluyan los nombres de los huesos con sus traslaciones y rotaciones. De forma automática el componente va a buscar los huesos en los que coincidan los nombres (por lo que es importante tener en cuenta que deben tener el mismo número de huesos y nombres)p, y establecerá las transformaciones correspondientes a estos huesos.

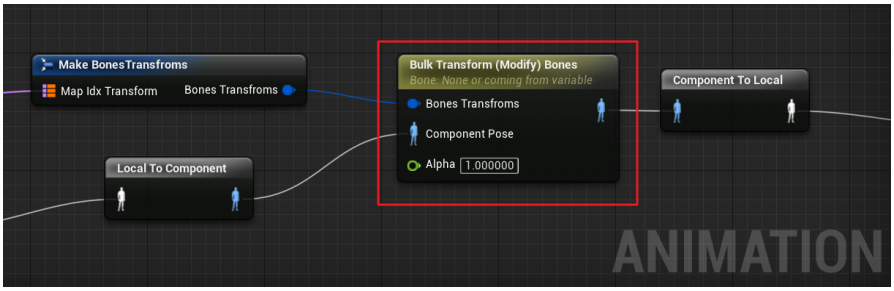



**Figura 4.11:** Ejemplo de empleo del componente *Bulk Transfor* el plugin AnimNode.

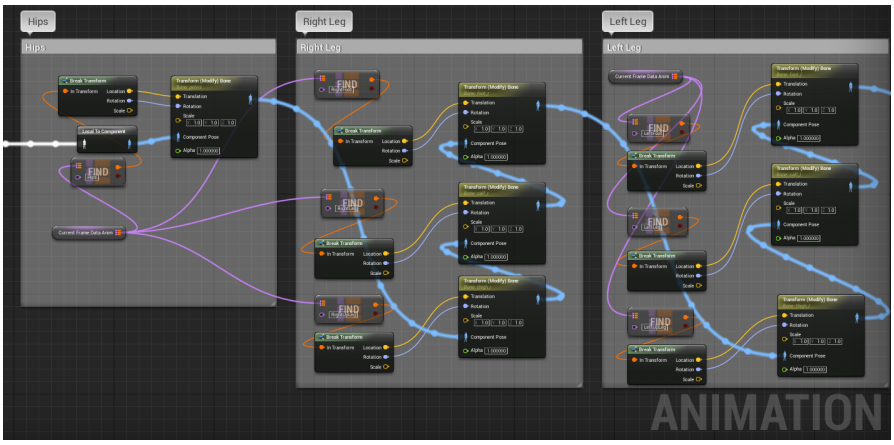



**Figura 4.12:** Ejemplo de uso de las transformaciones si se prescinde de este plugin.

[12]https://github.com/3dperceptionlab/unrealrox/tree/master/Plugins/AnimNode

Esto resulta muy útil para procesar la gran cantidad de huesos, 59 en nuestro caso, y asignarle los valores correspondientes. Para entender mejor su funcionamiento, podemos observar la Figura 4.12, para cada uno de los huesos es necesario establecer una transformación independiente, seleccionando el hueso uno a uno para cada operación. Acto seguido, incluir en esta transformación los valores de traslación y rotación del hueso en cuestión. La cantidad de componentes a emplear es considerablemente superior, por lo que el empleo de este plugin provoca una disminución del tiempo de desarrollo en este aspecto.

## 4.4. Control de la animación

Una vez establecidos todos los componentes en escena, todas las dependencias y la lógica necesaria se ha incluido en los componentes, es necesario asociar la animación con el esqueleto. Para ello es necesario crear un blueprint de animación a partir de nuestra clase Player BP, en cuyo interior incluiremos las instrucciones necesarias para asignar los valores del objeto TMap a los huesos del personaje.

Para comenzar, es importante conocer una restricción que Unreal presenta en este tipo de desarrollos. Cuando se esta empleando un blueprint de animación, este componente no puede acceder de ninguna forma a los datos almacenados en otro objeto, ni si quiera obteniendo su referencia, como hacíamos anteriormente. En este caso se procede a hacer llegar a este blueprint los valores deseados, esto se realiza desde la pestaña llamada *Event Graph*. En ella se obtiene el valor de la variable visible declarada en el Player BP y se asigna a otra variable del mismo tipo creada en el componente. El componente *Event Blueprint Update Animation* se ejecuta en cada uno de los frames de la animación, por tanto, el valor de la variable indicada se actualizará en cada frame.

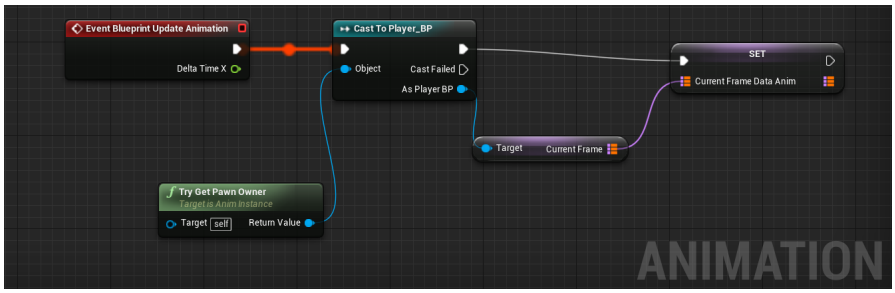



**Figura 4.13:** Método que establece el valor de TMap en el blueprint de la animación.

Ahora que este valor ya esta disponible para ser utilizado, se procede a asignar estos valores en las traslaciones y rotaciones de los huesos. Lo primero es seleccionar la técnica que vamos a emplear para aplicar estas transformaciones. En el caso de este proyecto, voy a emplear el plugin AnimNode ya que aporta una solución mucho más limpia y sencilla en lugar de aplicar las transformaciones por separado. La variable en la que se almacena el TMap debe ser convertida en una transformación, ya que el componente Bulk solo acepta este tipo de datos. En el componente pose, en este caso no es necesario incluir ninguna pose, ya que se sobrescribirá en cada ejecución, y como valor

alpha se establece por defecto a 1.

En los detalles del componente se pueden apreciar tres secciones separadas: traslacion, rotación y escala. Esta última no es necesario en este caso, por tanto en el modo en que se aplica se ha seleccionado ignorar. En las otras dos, en función del método empleado para grabar las acciones y exportarlas, sera preciso establecer un modo u otro, en este caso la configuración es la siguiente: Reemplazar existente y establecer en el espacio del componente.

Estableciendo esta configuración vamos a poder reproducir la animación frame a frame pulsando la letra K, o la letra establecida en cada caso para esta acción. Si al reproducir la animación los huesos no se establecen en la posición y rotación correctas, es posible que los valores indicados en el párrafo anterior no sean los correctos, el espacio y el modo en que se aplican estas transformaciones son la principal causa de error en estos casos. También es importante tener en cuenta los nombres de los huesos, ya que recordemos que el plugin AnimNode precisa que tanto el TMap como el esqueleto tengan los mismos nombres en sus huesos y la misma jerarquía de huesos.

## 4.5. Generar datos a partir de las grabaciones y el etiquetado

La generación de datos se trata del último paso para la construcción de nuestro dataset. Esta se realiza por medio del proyecto que tiene como nombre UnrealROX (4.2), desarrollado por la Universidad de Alicante. Va a servir como medio de ejecución para todas las grabaciones tomadas anteriormente y generar las secuencias de imágenes correspondientes. Para ello es necesario tener en cuenta los aspectos más importantes que involucran el uso de esta herramienta, al igual de los cambios pertinentes a esta para su completa funcionalidad.

Anteriormente se precisaba de la asignación de una tecla concreta del teclado para activar la ejecución de la animación. Con el uso de esta herramienta, esta actuación es ligeramente diferente, puesto que esta ya no se controla de forma manual. La ejecución del modo reproducción de UnrealROX es el encargado de hacer las llamadas pertinentes para el control de la animación. El acceso a los datos de traslación y rotación también sufren un ligero cambio, ya que no se incluye en el blueprint del actor. En las secciones siguientes se explican con mayor detalle los puntos mencionados.

### 4.5.1. Actor

En lo que respecta a UnrealROX, se dispone de un actor con el nombre de ROXMannequinPawn. Este es similar al actor diseñado en la Sección 4.3, pero suprimiendo la lógica correspondiente a la obtención de los datos de traslación y rotación. Este actor contiene por defecto la malla y esqueleto incluidos en el proyecto base de Unreal Engine 4. Para una correcta funcionalidad con nuestras grabaciones, es necesario sustituir esta malla y esqueleto por los mismos que emplea Axis Neuron. Esto se debe a la distribución de nombres que emplean los esqueletos, es preciso disponer del mismo orden, nomenclatura, posiciones iniciales en los huesos y misma pose inicial.

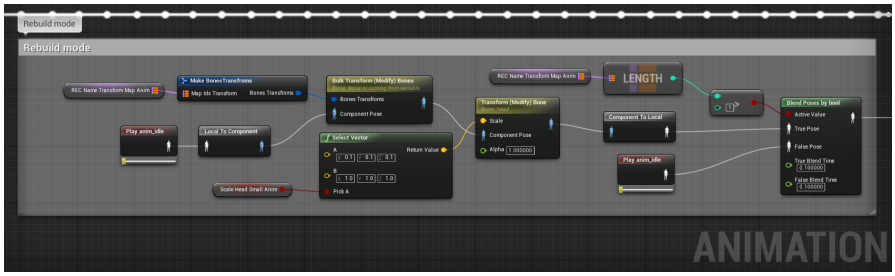



**Figura 4.14:** Sección en la animación del blueprint encargada de aplicar los cambios en el esqueleto del actor.

El blueprint de animación enlazado con el actor contiene la lógica que le permite tanto grabar, como reproducir grabaciones (como es el caso de este apartado). En la Figura 4.14 se pueden apreciar ciertas similitudes con la animación diseñada en la Sección 4.3, las cuales permiten obtener las transformaciones y establecer dichos cambios en su esqueleto. Estos datos se obtienen e incluyen en el actor por medio de C++, en lugar de blueprints como ocurría anteriormente.

### 4.5.2. Cámaras

Las reconstrucciones llevadas a cabo precisan de cámaras que tomen las instantáneas de cada uno de los frames de la animación. Se pueden localizar en cualquier punto de la escena, pero siempre apuntando directamente al actor o a la zona involucrada en la acción a desempeñar. En el actor de UnrealROX se incluye una cámara que esta posicionada en la cabeza del esqueleto. Se trata de una cámara empleada en desarrollos en primera persona, puesto que se esta trabajando con tercera persona se va a obviar esta cámara.

### 4.5.3. Tracker

Este componente es el orquestador principal de los componentes involucrados y de los datos recibidos por parte del plugin de BVH Player. En primer lugar, el componente asociado al plugin ejecuta el método play que lee los datos del fichero BVH (Sección 4.3). En cada ejecución del proceso asociado a la reconstrucción de la animación, acción correspondiente al ROXTracker, se hace una llamada al método Next del plugin. Estos datos son recogidos e inyectados desde el tracker al actor en escena, ya que posteriormente va a ser empleado por la animación del blueprint.

En cada uno de los frames esta animación va a recoger estos datos y aplicar las transformaciones necesarias en los huesos del esqueleto. Todos estos movimientos se pueden apreciar por medio de las cámaras insertadas en la escena. Anteriormente se hacía referencia al tracker como un orquestador, esto se debe a los datos que son necesarios aportar en los detalles de este componente. Dividido en secciones, disponemos de los siguientes datos a incluir:

- ROXTracker: Encontramos las características principales del componente, encargados del modo de actuación o directorios de exportación de los datos reconstruidos. Para emplear el método de reconstrucción se debe desmarcar el campo cuyo nombre corresponde con Record Mode.

- JSONManagement: Establece los directorios para los ficheros JSON empleados por el componente.
- Recording: Es el apartado correspondiente a la grabación del actor y de sus movimientos. A pesar de no emplear este punto, es preciso configurar sus variables con las referencias al actor y a las cámaras empleadas.
- Playback: Es el punto encargado de la reconstrucción de la animación y la generación de los datos. Se pueden establecer los directorios de exportación de datos, el tamaño de las imágenes (por defecto 1920x1080) y una referencia al plugin incluido. Esta referencia es necesaria para permitir que UnrealROX emplee los métodos de lectura y ejecución establecidos anteriormente.

## 4.6. Resumen

En esta sección se ha comenzado hablando sobre UnrealROX y el método por el cual ha sido empleado. El propósito ha sido poder generar de forma automática, un dataset sintético por medio de herramientas como las que facilita este proyecto. Además la incorporación de diversos plugins a la actuación de UnrealROX, ha permitido obtener a raíz de unos ficheros con datos de grabaciones, la información relevante para reproducir acciones sobre nuestro esqueleto.

Sin olvidarnos por supuesto de la técnica cuyo nombre es: retargeting. Uno de los puntos más preocupantes y a la vez importantes para establecer las transformaciones en el esqueleto ha sido este factor. El conjunto de huesos a emplear, debe ser idéntico, tanto en posición inicial, en los offsets y en los nombres de estos huesos. Si no se cumplen estas condiciones, lo más probable es que el movimiento de nuestro actor no se ajuste a la animación real.

Por último, es preciso remarcar el conjunto de interacciones entre los objetos presentes de la escena. Cada uno, desde el actor hasta el tracker, desempeña un papel fundamental en el desarrollo de la generación de datos. Nuestro tracker hace las veces de orquestador: indica cuando ejecutar el siguiente frame, asigna las transformaciones al esqueleto y controla cuando se debe reproducir y cuando grabar (en este caso, solo se emplea la reproducción). Nuestro actor recibe estos datos y su animación transforma su esqueleto para ajustarse a esta información, pero previamente han sido obtenidos por medio del plugin desarrollado en este capítulo. Una vez llegados hasta este punto, solo resta generar de forma automática, cuantos datos se deseen a partir de los generados a mano.

# Capítulo 5

# Resultados y experimentos

*En el siguiente capítulo se van a mostrar los resultados obtenidos en el uso de los diferentes plugins 5.2 y del resultado final de generar datos mediante ficheros BVH 5.3.*

## 5.1. Introducción

Los resultados constituyen un punto de gran importancia, ya sean resultados de pruebas correctas o incorrectas. Puesto que estas últimas permiten comprender cuales han sido los posibles errores y determinar una nueva estrategia para obtener el resultado deseado. Por ello, se divide la experimentación en dos secciones diferentes: pruebas con los diferentes plugins empleados para leer los ficheros BVH (5.2) y pruebas de generación de datos con la herramienta UnrealROX (5.3).

## 5.2. Plugins empleados

Esta sección recoge los tres plugins empleados durante el Capítulo 4. Los tres fueron probados para la mismas tareas: leer el fichero BVH, tratarlo y ejecutar la animación pero frame a frame de forma controlada por un operador, en lugar de estar lo el propio Unrela Engine.

### 5.2.1. Perception Neuron

Este es el plugin oficial ofrecido de forma gratuita por los desarrolladores del Axis Neuron [1]. Lo cierto es que resulta muy fácil de importar y usar, puesto que esta pensado para trabajar con este tipo de ficheros de animación. Pero se encuentran diferentes problemas a la hora de usarlo:

- Nombre del hueso padre en el esqueleto: Este es el principal problema que se encuentra al usar este plugin. Cuando se incluye la animación y se intenta asignar un esqueleto diferente, falla debido a la diferencia entre los nombres del hueso root de la animación (Figura 5.1). Unreal Engine 4 compara este hueso padre tanto de la animación como del esqueleto, y trata de emparejarlos a través de su nombre, pero al ser diferente aparece un error.

  La solución más sencilla de implementar para este problema (aunque a gran escala deja de ser factible) es modificar por medio de blender esta propiedad, es decir el nombre del hueso padre, para cada una de las animaciones. Es decir, por ejemplo si nuestro esqueleto dispone de este hueso con el nombre: root, modificar el correspondiente hueso en la animación con el mismo nombre.

[1] https://neuronmocap.com/content/unreal-plugin

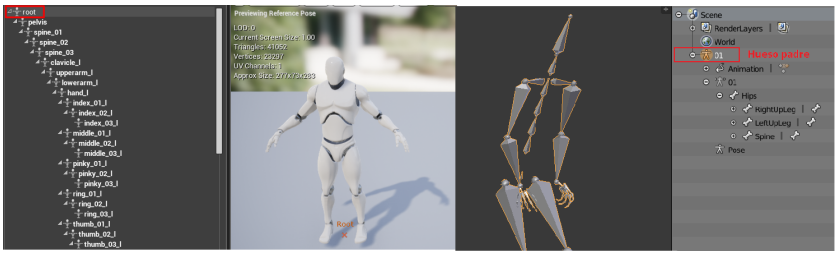


**Figura 5.1:** Árboles de huesos correspondientes a diferentes esqueletos: Mannequin Unreal Engine 4 (a la izquierda) y Axis Neuron Squeleton (a la derecha).

- Retargeting del esqueleto: Posiblemente este se trate del punto más importante para emparejar un esqueleto con una animación asociada a un esqueleto diferente al que estamos empleando. Lo más probable es que aparezca en la gran cantidad de veces que trabajemos con Unreal Engine 4 y animaciones. El problema ocurre cuando se intenta aplicar la animación de un esqueleto a otro emparejando los huesos por nombres, al igual que ocurría con el hueso root del apartado anterior. Pero al ser diferentes las los esqueletos, también lo son los nombres (la mayoría de veces).

  Por tanto la solución pasa por emplear una técnica conocida como retargeting [2]. En la cual se ajusta cada uno de los huesos de los diferentes esqueletos entre sí, para que el editor pueda generar una animación correcta, empleando para ello estas relaciones creadas por medio del retargeting.


**Figura 5.2:** Captura del proceso de retargeting para la relación de huesos entre dos esqueletos distintos.

- Funcionalidades limitadas: Una vez resueltos los puntos anteriores, resta probar que funciona correctamente, y así es. Pero el objetivo era obtener una reproducción frame a frame controlada a voluntad. Lo que ocurría realmente con este plugin fue la posibilidad de ejecutar la animación sobre el esqueleto, en una sola dirección de tiempo y sin poder siquiera pausar la animación. Estas restricciones

[2] https://docs.unrealengine.com/en-US/Engine/Animation/RetargetingDifferentSkeletons/index.html

no permitían obtener la solución deseada, por ello se decidió prescindir de este plugin.

### 5.2.2. Perception Neuron Template

A consecuencia de los experimentos llevados a cabo con el plugin anterior, se tomo la decisión de buscar otro desarrollo que se ajustara aún más al objetivo planteado. Por ello, se tomo el template siguiente, que permitía no solo lo anterior, sino también reproducir, pausar y rebobinar la animación. Este plugin está desarrollado bajo la licencia del MIT [3].

Lo cierto es que esta herramienta tampoco permitía ejecutar a voluntad los frames por separado. Por tanto, en lugar de seguir buscando, se encamino el proyecto a desarrollar una solución propia para este cometido. A pesar de estar cerca, no fue posible ajustar este template a las necesidades concretas que se tienen, los problemas encontrados fueron los siguientes:

- Lógica compleja en el método Tick: Al tratar de modificar este template, se procedió a cambiar la lógica necesaria e incluir un método que permitiera ejecutar un único frame en la animación y acto seguido pausarla. Pero no era posible, ya que no había un control concreto de estos frames, puesto que al ejecutarlo en el tick, se ajustaba la animación a los frames del juego, sin importar cuantos frames se ejecutaran en cada segundo.

  Se consiguió obtener una solución casi completa, ya que se ajustaron los parámetros necesarios para controlar que solo se ejecutara un frame. Pero esto conllevaba a la perdida de varios frames al inicio de la animación. Sucedía esto, puesto que en el tick se realizaba en primera instancia una lectura de los datos del BVH, y posteriormente se reproducía la animación. Al tener toda la lógica en el tick fue casi imposible eliminar esta restricción.

- Gran acoplamiento de los componentes: Además se encontró un alto acoplamiento entre el actor, el character y el nivel en el que se encontraban. En el blueprint del nivel se asignaba una referencia al character empleado para poder asignar las traslaciones y rotaciones de la animación en él (Figura 5.3). Esto provoca grandes problemas cuando cambiamos de nivel, puesto que sería necesario incluir de nuevo estos bloques en el nuevo escenario.

  En el plugin de BVH Player se solucionó este acoplamiento empleando un player controller, para así incluir este código en su interior. Simplemente fue necesario obtener la referencia del character en su interior y establecer dicho player controller para ese actor.

[3] https://forums.unrealengine.com/development-discussion/animation/56791-perception-neuron-template

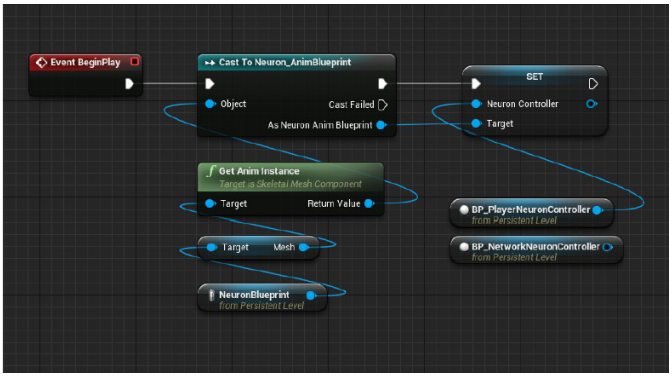



**Figura 5.3:** Lógica incrustada en el level blueprint del template Perception Neuron.

## 5.3. Generación de datos

La generación de datos se trata el último paso para obtener una dataset sintético a partir de animaciones reales tomadas por nosotros mismos. Al realizar este proceso, lo primero que se intentó fue emplear el esqueleto por defecto que incluye Unreal Engine 4. Pero al igual que ocurría en la sección anterior, esto produjo errores a la hora de relacionar los huesos del esqueleto y de la animación. El resultado fue una generación de imágenes que no correspondían con los movimientos correctos (Figura 5.4). Para poder llegar a este punto, fue necesario modificar el fichero BVH cambian los nombres de los huesos para que tanto estos como los del esqueleto coincidieran. Lo cierto es, que esta tarea es un tanto descabellada, pero simplemente se trataba de una prueba para comprobar el funcionamiento de la herramienta.

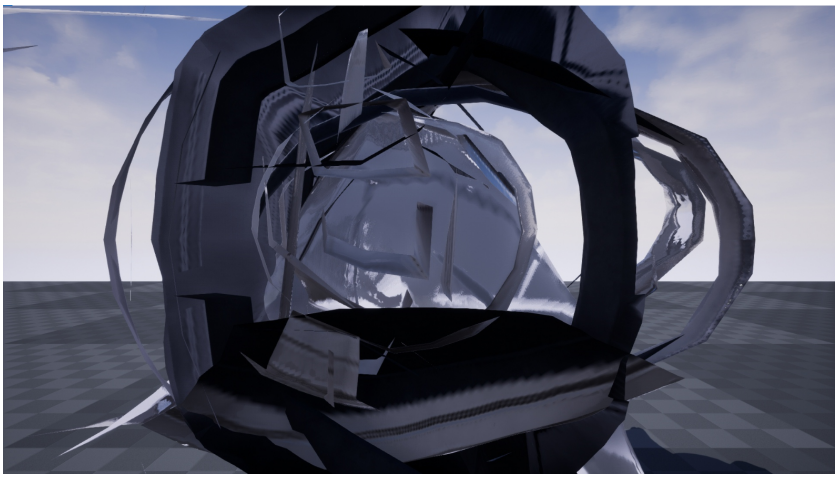


**Figura 5.4:** Salida obtenida por la cámara que graba la acción reproducida en el actor. Los huesos del esqueleto y animación no se corresponden, por ello la secuencia de acciones no se realiza correctamente.

La forma planteada para solventar este problema, pasa por adecuar el esqueleto correspondiente, en este caso el del Axis Neuron, para ser empleado por el Tracker y el BVH Player, creando además una animación. Pues bien, esto resulta más sencillo de lo que parece a simple vista, basta con cambiar la malla del actor y crear una nueva animación de blueprint para esta nueva malla. En su interior (blueprint de la animación)

incluimos la lógica que había anteriormente, y nos permitía actualizar las posiciones y rotaciones de los huesos del esqueleto. Pero esta vez al tratarse del correcto, nuestro actor ejecutará la acción correspondiente correctamente, como resultado tendremos en siguiente escenario (Figura 5.5).

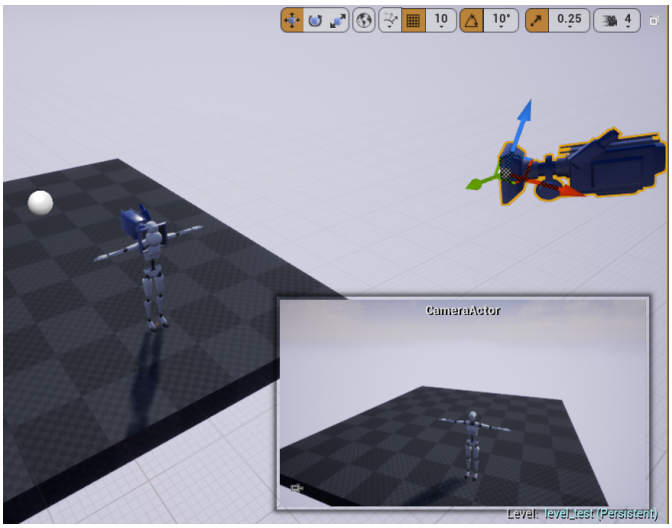



**Figura 5.5:** Escena del resultado final de emplear el Tracker con una cámara que graba al actor, siendo este la malla y esqueleto del Axis Neuron, cuya relación de huesos con la animación se hace correctamente debido a la jerarquía de nombres.

## 5.4. Resumen

De estos experimentos, se puede ver la cantidad de problemas que surgen cuando se trata de emplear mallas y esqueletos junto con animaciones en Unreal Engine 4. Pero si a esta fórmula le sumamos que tratamos de trabajar con animaciones y esqueletos diferentes... No puede acabar nada bien. Por suerte, como se ha tratado en el capítulo, basta con emplear retargeting cuando sea preciso o usar estructuras de huesos correctas para evitar que ocurra este tipo de cosas. Los errores de esta índole siempre están al acecho, afortunadamente tienen solución.

# Capítulo 6

# Conclusión

*Este capítulo recoge una pequeña vista del trabajo realizado. Se divide en varias partes: una conclusión personal 6.1 y puntos clave del proyecto 6.2.*

## 6.1. Conclusión

Este trabajo de final de grado surgió como una propuesta para mejorar la calidad de vida de personas con cierto nivel de discapacidad. Para empezar, este siempre ha sido un tema muy motivador, puesto que pienso que la ingeniería se originó para ayudar o mejorar las condiciones actuales. Y así es, cada día esta ingeniería de la que hablo permite a multitud de personas en todo el mundo dotar de conocimientos para desarrollar, construir o simplemente plantear una idea, que quizás mañana cambie el mundo. Puede que algunos me tachen de un tanto sensacionalista, pero yo prefiero pensar que se trata más de una realidad que de una sensación. Por supuesto, trato de reflejar estos factores en cada una de las frases o palabras que escribo, o en cada minuto que he dedicado a este proyecto.

Por un lado, se ha profundizado durante todo el documento sobre la creación de un dataset sintético robusto, que garantice que estas acciones se interpreten de forma correcta por el robot correspondiente, puesto que de ello va a depender que este robot realice un análisis efectivo y consiga segmentar la acción para comprender que esta ocurriendo. Tal y como se ha mostrado en los experimentos realizados, se ha conseguido aplicar satisfactoriamente una animación a un esqueleto y generar a partir de estos una gran cantidad de datos extraídos de cámaras situadas en la escena. Pero para alcanzar este objetivo se ha tenido que pasar por pasos como: el retargeting, desarrollo de un plugin para la integración de los BVH en Unreal Engine 4, y adaptar UnrealROX para emplearlo como director de orquesta y generador de datos.

No nos podemos olvidar del análisis que se realizó para comprender de que forma se mueven los humanos sobre determinadas acciones. Esto facilitó la comprensión de este tipo de tareas, para replicarlas lo más similares posibles mediante el traje compuesto por neuronas, correspondiente al Axis Neuron.

Para terminar, y aunque no se haya abordado en este trabajo de final de grado, hay un aspecto que ha sido nombrado en múltiples ocasiones y es de gran interés comentarlo, ya que es la finalidad última del proyecto. Este es el uso de todos estos datos para el entrenamiento de redes neuronales que, en un caso ideal, permitirían a un robot reconocer las acciones que una persona dependiente esté realizando. De esta forma, se tendría un gran punto de partida para que un robot situado en una habitación con una

persona que padece algún tipo de discapacidad, pueda servirle de ayuda para que sea capaz de realizar multitud de actividades.

## 6.2. Puntos clave

Los puntos clave del proyecto han sido los siguientes:

- Comprender el movimiento de los humanos en ciertas acciones.
- Replicar estos movimientos mediante el uso de un traje especializado.
- Etiquetar las animaciones grabadas mediante un script.
- Incluir estas animaciones y etiquetas en Unreal Engine 4 por medio de un plugin especial que permite leer y aplicar la información a un actor en concreto situado en la escena del nivel.
- Emplear UnrealROX junto con el plugin anterior para reproducir esta animación y aplicar las transformaciones de forma automática.
- Generar un dataset sintético y estructurado que constituirá el pilar base para el análisis de las acciones.

# Apéndice A

# Script de etiquetado de acciones

*En este apéndice se puede encontrar una breve muestra del código empleado para el etiquetado de acciones grabadas por medio del hardware y software con los nombres de Perception Neuron y Axis Neuron respectivamente. El código completo del script se puede encontrar en el repositorio empleado para este proyecto.*

## A.1. Menú principal

Las opciones presentadas son diversas, las mas importantes de destacar son la posibilidad de cambiar de directorio y fichero por medio del propio script, el etiquetado de la animación y el muestreo de este fichero JSONs generado.

```
1  def menu():
2    print("Selecciona la opcion que deseas realizar (1-8)\n" +
3    "\t1. Ver fichero cargado.\n" +
4    "\t2. Seleccionar nuevo fichero.\n" +
5    "\t3. Ver datos del fichero.\n" +
6    "\t4. Ver ruta de video.\n"
7    "\t5. Cambiar ruta de video.\n"
8    "\t6. Comenzar analisis de datos.\n" +
9    "\t7. Ver datos analizados.\n"
10   "\t8. Salir.")
11
12   op = int(input())
13   return op
```

## A.2. Selección de directorio y fichero

Estas dos opciones se encuentran disponibles para poder modificar los directorios de los diferentes archivos sin necesidad de recompilar el código, ya que se presupone que en proyectos más grandes estas clases es muy probable que sean mucho más voluminosas.

```
1  def opciones(path_video, name_file, data, op, dir_fichero):
2    if op <= 8 and op >= 1: # Lista directorio actual
3      if op == 1:
4        ...
5      elif op == 2: # Seleccionar fichero de la lista de ficheros en el
           directorio
6        bucle = True # Variable de salida del bucle
7
8        while bucle:
```

```
9           result = ls(path_video,numeracion=True,excluir='json') # Obtiene un
                ls del directorio
10          print("Escribe el nombre del fichero deseado (c cancelar):")
11          new_name = input() # Nombre de fichero seleccionado
12
13          if new_name in result: # Comprueba si el nombre del fichero se
                encuentra
14            name_file = new_name
15            print("\033[0;32m" + "Archivo modificado." + '\033[0;m')
16            continuar()
17            bucle = False
18          elif new_name == 'c': # Permite salir sin realizar cambios
19            bucle = False
20          else: # Error
21            print("\033[0;31m" + "El archivo no se ha encontrado, por favor
                  vuelve a intentarlo o cancela la accion (c).\n" + '\033[0;m')
22
23        return 1, name_file, path_video, data
24      elif op == 3:
25        ...
26      elif op == 4:
27        ...
28          elif op == 5: # Cambiar ruta del directorio actual
29        bucle = True # Variable de salida del bucle
30
31        while bucle:
32          print("Escribe el nombre de la ruta deseada:")
33          new_path = input() # Nombre de fichero seleccionado
34
35          if is_path_exist(new_path): # Comprueba si la ruta existe
36            if new_path[len(new_path) - 1] != '/': # Incluye el caracter / si
                  no existe al final del string
37              new_path += '/'
38            path_video = new_path
39            print("\033[0;32m" + "Ruta modificada." + '\033[0;m')
40            continuar()
41            bucle = False
42          elif new_path == 'c': # Permite salir sin realizar cambios
43            bucle = False
44          else:
45            print("\033[0;31m" + "La ruta no se ha encontrado, por favor vuelve
                  a intentarlo o cancela la accion (c).\n" + '\033[0;m')
46        return 1, name_file, path_video, data
47      elif op == 6:
48        ...
49      elif op == 7:
50        ...
```

## A.3. Etiquetado de la animación

Esta parte del código permite obtener a partir del fichero BVH el número de frames almacenados en dicha grabación. Posteriormente se procede a analizar de forma manual frame a frame, etiquetando por medio de un menú cada uno de los diferentes movimientos a partir de una lista de clases definidas. Cabe destacar que en ciertas ocasiones, el movimiento en el frame $t_0$ es igual (dirección y movimiento) al frame $t_1$, por tanto es posible omitir hasta el momento en el que finaliza esta acción

```
...

  elif op == 6:
    try:
      data_fich = leer_clases(dir_fichero).split('-') # Obtiene las clases determinadas
      data = analisis(path_video, name_file, data_fich) # Genera el analisis
      data = cambiar_repetidos(data) # Elimina valores repetidos
    except: # Excepcion en la apertura del fichero
      data = None
      data_fich = None
      print("\033[0;31m" + "El archivo no se puede abrir, compruebe el nombre de la ruta y/o del fichero.\n" + '\033[0;m')
      print_error(sys.exc_info())
      continuar()
    ...
    return 1, name_file, path_video, data

...

def analisis(path_video, name_file, data_fich):
  data = []

  if WINDOWS: # Conversion de ruta para windows
    path_video = parse_path_windows(path_video)

  with open(path_video + name_file) as f: # Se abre el fichero BVH
    mocap = Bvh(f.read()) # Empleamos la clase bvh para leer el fichero

    n_frames = next(mocap.root.filter('Frames:')).value[1] # n_frames = numero de frames

    i = 1
      while i < int(n_frames) + 1: # Este bucle se repite hasta que se superen los frames totales
      print("\033[0;32m" + "Frame: " + str(i) + '\033[0;m')
      sub_data, salto = menu_analisis(data_fich) # Imprime un menu de seleccion

      if salto: # Permite saltar frames para indicar una misma accion consecutiva
        print("\033[0;33m" + "Indica el frame hasta el que se repite la acción anterior:" + '\033[0;m')
        try:
          frame_final = int(input())
          for x in range(i, frame_final + 1): # Genera el valor anterior x veces
          data_ = [0]
          data.append(data_)
          i += 1
        except: # Permite controlar el valor introducido para que siempre sea numérico
          print("\033[0;31m" + "Los valores introducidos deben ser numericos.\n" + '\033[0;m')
          print_error(sys.exc_info())
      else:
        data.append(sub_data)
      i += 1
```

```
49    return data
50
51  ...
52
53  def menu_analisis(data_fich):
54    data_fich = iter(data_fich) # Iterador sobre el fichero bvh
55    next(data_fich) # Se obtiene el siguiente valor
56    salto = False # Condicion que determina el salto de frames repetidos
57    data = []
58    print("\033[0;33m" + "Acción de nivel 1 (si es la misma 0 o -1):" + '
          \033[0;m')
59    print(next(data_fich))
60    n_1 = int(input()) # Obtiene la entrada del usuario del nivel 1 (actividad:
           Estatico)
61    if n_1 != 0 and n_1 != -1: # Determina si es preciso un salto o no (-1 ==
          salto)
62      print("\033[0;33m" + "Acción de nivel 2:" + '\033[0;m')
63      print(next(data_fich))
64      n_2 = int(input()) # Obtiene la entrada del usuario del nivel 2 (
            actividad: Caminar)
65      print("\033[0;33m" + "Acción de nivel 3:" + '\033[0;m')
66      print(next(data_fich))
67      n_3 = input() # Obtiene la entrada del usuario del nivel 3 (actividad:
            Mover pierna derecha)
68
69      n_3 = n_3.split(' ')
70      n_4 = []
71
72      try:
73        int(n_1)
74        int(n_2)
75        for x in n_3:
76        n_4.append(int(x))
77      except:
78        print("\033[0;31m" + "Los valores introducidos deben ser numericos.\n"
              + '\033[0;m')
79        print_error(sys.exc_info())
80
81      # Los datos se incluyen en un array para ser tratados posteriormente
82      data.append(n_1)
83      data.append(n_2)
84      data.append(n_4)
85    else:
86      if n_1 == 0:
87        data.append(0)
88      else:
89        salto = True
90
91    return data, salto
92
93  ...
```

## A.4. Generación del fichero JSON

Para reconstruir los datos almacenados es necesario recorrer los tres niveles de abstracción por separados y asignar los frames correspondientes a estos niveles. Estos

cálculos se hacen por separado en diferentes módulos, aunque por simplificar, unicamente se ha presentado el cálculo del nivel 1.

```
...

    elif op == 6:
    ...
    try:
      json = generar_json(data, data_fich) # Genera el fichero JSON
      new_name_file = path_video + name_file # Crea el nombre del fichero
      exportar_json(json, new_name_file) # Busca la ruta y exporta el fichero
    except:
      print_error(sys.exc_info())
  return 1, name_file, path_video, data

...

def generar_json(data, data_fich):
  json = '{\n'
  data_fich = iter(data_fich) # Genera un iterador sobre el fichero
  next(data_fich) # Obtiene el siguiente elemento
  if data and data_fich: # Si se han encontrado datos procede a generar el
      archivo JSON
    print("\033[0;33m" + "Generando fichero JSON..." + '\033[0;m')

    # En este instante es preciso recorrer los 3 niveles de abstraccion
    data_l1 = next(data_fich).split('\n')
    data_l2 = next(data_fich).split('\n')
    data_l3 = next(data_fich).split('\n')
    data_l1 = data_l1[1:len(data_l1) - 1]
    data_l2 = data_l2[1:len(data_l2) - 1]
    data_l3 = data_l3[1:len(data_l3) - 1]

    # Primer nivel
    i = 0
    for x in data_l1: # Se realiza el parseo en el nivel 1
      data_aux = []
      data_l1_aux = x.replace(' ', '').split('.')
      for y in data:
        if y[0] == int(data_l1_aux[0]):
          data_aux.append([y[1], y[2]])
      if len(data_aux) != 0:
        json += '\t"' + data_l1_aux[1] + '": {\n'

        # Se realiza el parseo en el nivel 2 y 3
        json += nivel_2(data_aux, data_l2, data_l3) # Esto devuelve los datos
            de l2 y l3 ya tratados
        json += '\t},\n'
      i += 1
  json = json[:-2] + '\n' + '}'
  return json

...
```

# Apéndice B

# Plugin BVH Player en Unreal Engine 4

*En este apéndice se puede encontrar una breve muestra del código empleado para la carga y extracción de los datos almacenados en los ficheros BVH, y de su reproducción en el entorno gráfico.*

## B.1. Métodos y variables

Conjunto de variables y métodos empleados para almacenar y leer los datos del fichero BVH, para poder trabajar con estos durante la reconstrucción de la animación. En concreto, los huesos se almacenan en el componente TArray<FString> Bones, y la secuencia de transformaciones en TArray<TMap<FName, FTransform>> REC_Map.

```
1  public:
2    ...
3    bool Play(FString BVHFileName, bool bPlayEndless, USkeletalMeshComponent *
         Mesh, FName CustomBoneName);
4    TMap<FName, FTransform> Next();
5
6    FORCEINLINE int GetNumFrames() const
7    {
8    return REC_Map.Num();
9    }
10
11 private:
12   ...
13   TArray<FString> Bones;
14   FVector Translation;
15   FVector OffsetTranslation;
16   FRotator Rotation;
17
18   FQuat CalculateQuat(float XR, float YR, float ZR);
19
20 public:
21   ...
22   // Variables con los datos del frame
23   UPROPERTY(BlueprintReadWrite)
24   TMap<FName, FTransform> CurrentFrameData;
25   TMap<FName, FVector> Offsets;
26   int Currentframe = 0;
27
28 protected:
29   UPROPERTY(BlueprintReadWrite)
30   TMap<FName, FTransform> REC_NameTransformMap;
```

```
31   TArray<TMap<FName, FTransform>> REC_Map;
```

## B.2. Métodos blueprint

En este código se aprecia los componentes creados para establecer las clases C++ en blueprints, y poder acceder a estar por medio del editor de blueprints. El fin es mantener las clases en C++ con toda o casi toda la lógica necesaria, y establecer nuestras clases blueprint con llamadas a las anteriores.

```
1  ...
2    // Funciones de Play y Next accesibles desde blueprint
3    UFUNCTION(BlueprintCallable, Category = "Perception Neuron", meta = (
         DisplayName = "Play BVH File", ToolTip = "Play a local stored BVH file.
         "))
4    static bool NeuronPlay(ABVH_Player* player, FString BVHFileName, bool
         bEndless, bool bReference, bool bDisplacement, USkeletalMeshComponent *
         Mesh, FName CustomBoneName = FName(TEXT("None")));
5
6    UFUNCTION(BlueprintCallable, Category = "Perception Neuron", meta = (
         DisplayName = "Next Frame BVH File", ToolTip = "Next Frame BVH file."))
7    static TMap<FName, FTransform> NeuronNext(ABVH_Player* player);
8  ...
```

## B.3. Lectura de fichero y extracción de información

Método encargado de leer el fichero BVH y cargar en diferentes estructuras de datos, la información correspondiente a los huesos, offsets de estos, duración de la animación y trasformaciones por frame en la grabación con dicho esqueleto.

```
1  // BVH Player
2  bool ABVH_Player::Play(FString BVHFileName, bool bPlayEndless,
       USkeletalMeshComponent *Mesh, FName CustomBoneName)
3  {
4    ...
5    // Load BVH file
6    ...
7    // Parse BVH file, get frame time and beginning of motion line part
8    ...
9    // Obtener TArray con un TMap<FName, FTransform>> por frame
10   bool fps_finded = false;
11
12   for (FString line : PlayerMotionLines){
13     // Primer hueso
14     if (line.Contains(TEXT("ROOT"))) {
15       FString Left, Right;
16       line.Split(TEXT(" "), &Left, &Right);
17       Bones.Add(Right);
18     }
19     else if (line.Contains(TEXT("JOINT"))) { // Resto de huesos
20       FString Left, Right;
21       line.Split(TEXT(" "), &Left, &Right);
22       Bones.Add(Right);
23     }else if (fps_finded) { // Solo se ejecuta cuando acaba de analizar la
           jerarquia de huesos
```

```
24        TArray<FString> line_slited;
25        line.ParseIntoArray(line_slited, TEXT(" "), false);
26
27        FQuat RefQuat(FQuat::Identity);
28        if (Mesh != nullptr && Mesh->SkeletalMesh != nullptr){
29          const FReferenceSkeleton& RefSkeleton(Mesh->SkeletalMesh->RefSkeleton
              );
30          int32 RefBoneIndex = Mesh->GetBoneIndex(CustomBoneName);
31          while (RefBoneIndex != INDEX_NONE){
32            RefQuat = RefSkeleton.GetRefBonePose()[RefBoneIndex].GetRotation()
                * RefQuat;
33            RefBoneIndex = RefSkeleton.GetParentIndex(RefBoneIndex);
34          }
35        }
36
37        for (int i = 0; i < line_slited.Num() - 1; i += 6) {
38
39          // Translation
40          float X = FCString::Atof(*line_slited[i]) - Offsets[FName(*Bones[i /
              6])].X;
41          float Y = FCString::Atof(*line_slited[i + 1]) - Offsets[FName(*Bones[
              i / 6])].Y;
42          float Z = FCString::Atof(*line_slited[i + 2]) - Offsets[FName(*Bones[
              i / 6])].Z;
43
44          Translation = FVector(Y, Z, -X);
45
46          // Rotation
47          float XR = FCString::Atof(*line_slited[i + 3]);
48          float YR = FCString::Atof(*line_slited[i + 4]);
49          float ZR = FCString::Atof(*line_slited[i + 5]);
50
51          FRotator Rotation(ZR, YR, XR);
52          UKismetSystemLibrary::PrintString(GetWorld(), Rotation.ToString());
53          UKismetSystemLibrary::PrintString(GetWorld(), Translation.ToString())
              ;
54
55
56          FTransform trans;
57          trans = FTransform(Rotation, Translation, FVector(1,1,1));
58          REC_NameTransformMap.Add(FName(*Bones[i / 6]), trans);
59        }
60        REC_Map.Add(REC_NameTransformMap);
61        REC_NameTransformMap.Empty();
62      }else if (line.Contains(TEXT("Frame Time:"))) { // Si se han analizado
          todos los huesos, paso a analizar las translaciones y rotaciones
63        fps_finded = true;
64      }
65      else {
66        if (line.Contains(TEXT("OFFSET"))) {
67          FString Left, Right, OF1, OF2, OF3;
68          line.Split(TEXT(" "), &Left, &Right); // Aqui almacenamos la cadena "
              OFFSET"
69          Right.Split(TEXT(" "), &OF1, &Right); // Aqui almacenamos el primer
              offset
70          Right.Split(TEXT(" "), &OF2, &Right); // Aqui almacenamos el segundo
              offset
71          Right.Split(TEXT(" "), &OF3, &Right); // Aqui almacenamos el tercero
              offset
72          OffsetTranslation = FVector(FCString::Atof(*OF1), FCString::Atof(*OF2
              ), FCString::Atof(*OF3));
```

```
73            Offsets.Add(FName(*Bones[Bones.Num() - 1]), OffsetTranslation);
74          }
75        }
76      }
77  ...
78  }
```

## B.4. Obtención de los datos del siguiente frame

Este método se llama cada vez que pulsamos la tecla asignada para pasar al siguiente frame de la animación, o cuando el objeto Tracker realice esta invocación. Accede al componente que contiene la secuencia de traslaciones y rotaciones, y lee la siguiente posición, puesto que el tamaño de este es igual a la duración de la acción.

```
1  TMap<FName, FTransform> ABVH_Player::Next() {
2    if (Currentframe < Frames) {
3      CurrentFrameData = REC_Map[Currentframe];
4    }
5
6    Currentframe++;
7    return CurrentFrameData;
8  }
9
10 TMap<FName, FTransform> ABVH_Player::NeuronNext(ABVH_Player* player) {
11   return player->Next();
12 }
```

# Bibliografía


[1] Xavier Puig, Kevin Ra, Marko Boben y col. «VirtualHome: Simulating Household Activities via Programs». En: *Computer Vision and Pattern Recognition (CVPR)*. 2018. URL: http://virtual-home.org/.

[2] Daniel Nyga y Michael Beetz. «Everything Robots Always Wanted to Know about Housework (But were afraid to ask)». En: *IROS* (2012), págs. 243-250. URL: https://robohow.org/_media/special/bib/nyga12actioncore.pdf.

[3] Yezhou Yang, Anupam Guha, Cornelia Fermuller y col. «Manipulation Action Tree Bank: A Knowledge Resource for Humanoids». En: *IEEE-RAS Intl. Conf. on Humanoid Robots* (2014). URL: https://www.cs.umd.edu/~aguha/publications/treeBankHumanoid.pdf.

[4] Bernhard Wymann, Christos Dimitrakakis, Andrew Sumner y col. «TORCS: The Open racing car simulator». En: (2015). URL: http://www.cse.chalmers.se/~chrdimi/papers/torcs.pdf.

[5] Eric Kolve, Roozbeh Mottaghi, Daniel Gordon y col. «AI2-THOR: An Interactive 3D Environment for Visual AI». En: *ArXiv e-prints* (2017). URL: https://arxiv.org/pdf/1712.05474.pdf.

[6] Alexey Dosovitskiy, German Ros, Felipe Codevilla y col. «CARLA: An Open Urban Driving Simulator». En: *CORL* (2017). URL: http://proceedings.mlr.press/v78/dosovitskiy17a/dosovitskiy17a.pdf.

[7] *Unreal Engine 4*. URL: https://www.unrealengine.com/en-US/what-is-unreal-engine-4.

[8] German Ros, Laura Sellart, Joanna Materzynska y col. «The SYNTHIA Dataset: A Large Collection of Synthetic Images for Semantic Segmentation of Urban Scenes.» En: *CVPR* (2016). URL: https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Ros_The_SYNTHIA_Dataset_CVPR_2016_paper.pdf.

[9] César Roberto De Souza, Adrien Gaidon, Yohann Cabon y col. «Procedural Generation of Videos to Train Deep Action Recognition Networks.» En: *CVPR*. 2017, págs. 2594-2604.

[10] Ankur Handa, Viorica Patraucean, Vijay Badrinarayanan y col. «Understanding real world indoor scenes with synthetic data». En: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, págs. 4077-4085.

[11] Tsung-Yi Lin, Michael Maire, Serge Belongie y col. «Microsoft coco: Common objects in context». En: *European conference on computer vision*. Springer. 2014, págs. 740-755.

[12] Jamie Shotton, Andrew Fitzgibbon, Mat Cook y col. «Real-time human pose recognition in parts from single depth images». En: *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*. Ieee. 2011, págs. 1297-1304.

[13] Justin Johnson, Bharath Hariharan, Laurens van der Maaten y col. «Infering and Executing Programs for Visual Reasoning». En: *ICCV* (2017). URL: http://www.cs.cmu.edu/~jeanoh/16-785/papers/johnson-iccv2017-reasoning.pdf.

[14] Pablo Martinez-Gonzalez, Sergiu Oprea, Alberto Garcia-Garcia y col. «UnrealROX: An eXtremely Photorealistic Virtual Reality Environment for Robotics Simulations and Synthetic Data Generation». En: *ArXiv e-prints* (2018). eprint: 1810.06936. URL: https://arxiv.org/abs/1810.06936.

[15] *Perception Neuron*. URL: https://neuronmocap.com/.

[16] *Python.org*. URL: https://www.python.org/.

[17] *Axis Neuron*. URL: https://neuronmocap.com/downloads.

[18] *Blender*. URL: https://www.blender.org/.

[19] *Google Colab*. URL: https://colab.research.google.com.

[20] *TensorFlow*. URL: https://www.tensorflow.org/.

[21] *Keras*. URL: https://keras.io/.

[22] *PyTorch*. URL: https://pytorch.org/.

[23] Brandon Wilson, Matthew Bounds y Alireza Tavakkoli. «A full-body motion calibration and retargeting for intuitive object manipulation in immersive virtual environments». En: *2016 IEEE 9th Workshop on Software Engineering and Architectures for Realtime Interactive Systems (SEARIS)*. IEEE. 2016, págs. 1-5.

[24] Alireza Tavakkoli. *Immersive Virtual Reality with Applications to Tele-Operation and Training*. Inf. téc. University of Houston-Victoria Victoria United States, 2016.